An Evaluation of Phonetic Spell Checkers

Victoria J. Hodge
Dept. of Computer Science,
University of York
Heslington
York, UK
YO10 5DD
vicky@cs.york.ac.uk

Jim Austin
Dept. of Computer Science,
University of York
Heslington
York, UK
YO10 5DD
austin@cs.york.ac.uk

Abstract

In the work reported here, we describe a phonetic spell-checking algorithm, Phonetex which integrates aspects of Soundex and its extension Phonix. It is designed to provide a phonetic component for an existing typographic spell checker. We increase the number of letter codes compared to Soundex and Phonix. We also integrate phonetic rules but use far less than Phonix which was designed for South African name matching or Rogers and Willett's Phonix extension which was designed for 17th century spellings as these includes many rules that are redundant in a contemporary word-based domain. We evaluate our algorithm by comparing it to phonetic spell checkers, Soundex and Editex and four benchmark spell checkers (Agrep, MS Word 97 & 2000 and UNIX 'ispell') using a list of phonetic spelling errors. We find that our approach has superior *recall* (accuracy) to the alternative approaches although the higher recall is at the expense of *precision* (number of possible matches retrieved). We intend to integrate it into an existing spell checker so the precision will be improved by integration thus high recall is the aim for our approach in this paper.

Keywords: Data Cleaning, Phonetic Spell Checker, Phonetic Code Generation.

1 Introduction

In this paper, we develop a phonetic spell checker - Phonetex. Phonetic errors are more difficult to correct than other spelling errors as they distort the misspelled word more than a single insertion, deletion or substitution error [7]. Therefore, we develop a phonetic spell checker intended to integrate with an existing typographic spell checker [6] and handle phonetic misspellings not covered by the typographic approach. Pfeifer et al. [8] evaluated phonetic error correction methods on proper names and recommended integrating a conventional n-gram approach with a phonetic component for optimum recall which endorses our combined approach [6]. There are various phonetic spell checkers currently used such as Soundex [7] which produces codes to represent words but the quality of matches is poor [13] due to the limited code permutations. The Phonix [5] extension of Soundex produces better quality matches and is intrinsically more flexible than merely converting words directly to phonetic codes due to the use of phonetic transformation rules. However, Gadd's implementation [5] incorporates a large number of phonetic rules for South African name matching and Rogers and Willett's Phonix extension [10] was designed for 17th century spellings so many rules are not relevant in a contemporary word-based spell checker. In

0	1	2	3	4	5	6
aeiouhwy	bpfv	cgjkqsxz	dt	l	mn	r

Table 1: Table showing the Soundex codes for each letter

0	1	2	3	4	5	6	7	8	9
aeiouy	bp	ckq	dt	lr	mn	gj	fpv	sxz	csz

Table 2: Table showing the Editex code groups. The groups are not disjoint as scores are calculated using equivalence groups. The groups reflect phonetic correspondences where letters that are pronounced similarly are grouped together so s and z are equivalent to x in group 8 and equivalent to c in group 9.

this paper, we describe and evaluate our phonetic approach, Phonetex, which aims for high recall. We envisage Phonetex being integrated with a typographic approach [6] so maximising precision is less essential. We just need to integrate candidate matches from the phonetic and typographic components to score and rank the words. We therefore revise the Phonix 4-character code approach by increasing the number of letter codes to maximise the word code permutations while maintaining phonetic similarity and reducing the number of phonetic rules to a core rule base to minimise comparisons and maximise speed.

Our approach is language independent; it just requires the sound-alike letter codes and rules to be generated prior to implementation. It is simple, flexible and produces high *recall* (retrieval accuracy). There are two fundamental issues for phonetic spell checkers, speed and accuracy. In this paper we concentrate on the latter. We describe and comparatively evaluate a selection of spelling methodologies that include phonetic matching. Some approaches such as Soundex [7] or Editex [13] have been designed specifically for phonetic spelling while other approximate matching techniques used in spelling checkers such as Agrep are suitable for phonetic matching. All algorithms match a query word against a stored lexicon (word list).

Soundex (see [7] for details) was developed early in the 20th century and maps the letters of the alphabet on to a series of numeric codes with the exception of the first letter of the word which is mapped to itself (see table 1 for the codes and figure 1 for the algorithm in pseudocode). Each word is encoded by concatenating the codes of its constituent letters, ignoring 0s and only indexing the first letter if adjacent word letters map to identical code values. In Soundex the word code is limited to four characters (first letter|0-6|0-6|0-6). If there are less than four non-zero characters then the word code is padded with 0s. Words are matched by comparing their respective Soundex codes and counting the number of matching and aligned phonetic code characters. Soundex retrieves many false positives during matching (see table 5) due to a large set of letters mapping to each letter code value and many of these false positives do not sound like the target word. Soundex also does not rank the retrieved matches, words either match or they do not; the only score available is a count of the number of matches in the word code (from 0 to 4).

Editex [13] is an integration of Soundex code mapping listed in table 2 and a revised Edit Distance algorithm detailed in figure 2. Editex was designed for name error correction. The revised Edit Distance [13] counts the number of insertions, deletions or substitutions

```
code[1] := word[1]; // where code and word are indexed from 1 to m
i := 2;
for n in 2 to lengthOf(word) {
  If Soundex(word[n]) == 0 then skip;
  ElseIf Soundex(word[n]) == Soundex(word[n-1]) then skip;
  Else {
      code[i] := Soundex(word[n]);
      i := i + 1;
  }
}
If code has less than three digits then pad with 0s.
Truncate code at 4 characters giving letter|digit|digit|digit.
```

Figure 1: Figure listing the Soundex algorithm in pseudocode. Skip jumps to the next loop iteration. The function Soundex() returns the appropriate code mapping for the letter.

$$
\begin{aligned}
&edit(0,0) = 0 \\
&edit(i,0) = edit(i-1,0) + d(s_{i-1}, s_i) \\
&edit(0,j) = edit(0,j-1) + d(t_{j-1}, t_j) \\
&edit(i,j) = min[edit(i-1,j) + d(s_{i-1}, s_i) \\
&\qquad\qquad edit(i,j-1) + d(t_{j-1}, t_j) \\
&\qquad\qquad edit(i-1,j-1) + r(s_i, t_j)]
\end{aligned}
$$

Figure 2: Figure listing the Editex algorithm taken from [13] where $s_1 s_2 ... s_i$ is the misspelling and $t_1 t_2 ... t_j$ is the target word from the lexicon.

required to convert the query word to each lexicon word in turn. In function $r(a, b)$ in figure 2, equivalent letters score 0, equivalent code group letters score 1 and dissimilar letters score 2. The function $d(a, b)$ is identical to $r(a, b)$ except if a is `h` or `w` and $a \neq b$ then $d(a, b)$ returns 1. The letter comparison value is assigned as a cumulative lexicon word score. The lexicon word with the lowest score is deemed the best match. Edit Distance can be slow for a large dictionary as it has $O(mn)$ running time as all m letters of the query word are compared to all letters of each of the n lexicon words in turn.

Agrep [11] is a fast sub-string matching algorithm that performs the edit distance scoring. It identifies the words in the lexicon that are at most k insertions, deletions or substitutions from the query word. Agrep essentially uses arrays of binary vectors and pattern matching, comparing each character of the query word in order, to determine the best matching lexicon word. The binary vector acts as a mask so only characters where the mask bit is set are compared, minimising the computation required. Agrep is not specifically aimed at phonetic matching but is an ideal comparative algorithm.

Our Phonetex methodology adopts the Phonix approach of combining Soundex-type codes with phonetic transformation rules to produce a phonetic code for each word. We studied the etymology of the English language detailed in the Concise Oxford English Dictionary (COED), spelling errors gathered from Internet News Groups and handwritten text and

0	1	2	3	4	5	6	7	8	9	A	B	C	D
aeiouhwy	b	d	f	gj	k	l	m	n	p	r	sz	t	v

Table 3: Table giving our codes for each letter.

ˆhough → *h5*	ˆrough → *r3*
ˆcough → *k3*	ˆtough → *t3*
ˆchough → *s3*	ˆenough → *e83*
ˆlaugh → *l3*	ˆtrough → *tA3*
ˆps → s	ˆwr → r
ˆpt → t	ˆkn → n
ˆpn → n	ˆgn → n
ˆmn → n	ˆx → z
sc(e|i|y) → s	1. (i|u)gh(¬a) → _ 2. gh → g
+ti(a|o) → s	gn$ → n
ph → f	gns$ → ns
1. c(e|i|y|h) → s 2. c → k	q → k
mb$ → m	+x → ks

Table 4: Table of the phonetic transformation rules in our system. Italicised letters are phonetic codes - all other letters are standard alphabetical letters. ˆ indicates 'the beginning of a word', $ indicates 'the end of the word' and + indicates '1 or more letters'. Rule 1 is applied before rule 2.

integrated the most salient rules from Phonix and aspell [2]. The Phonetex rule base encompasses the COED as well as the smaller dictionaries used in the evaluation here. We only included rules that apply generally to produce a generic core rule base so any rules that have frequent exceptions were not included. Phonetex also uses a larger number of letter codes than Soundex, Phonix or Editex. We use fourteen characters compared to just seven in Soundex, nine in Phonix and ten in Editex. All letter groups are disjoint in our approach. The letters are translated to codes indexed from 0-D in hexadecimal to allow a single character representation for each letter code to ensure only the required length word codes are produced. The increased number of code permutations prevents too many words mapping to each code so we can minimise false positive words returned as a match that in fact only match poorly. Note that we devised our rule base and code mappings prior to obtaining the phonetic spelling error dataset used in our evaluation in this paper.

In Phonetex, any applicable transformation rules are initially applied to the word. The phonetic transformation rules are given in table 4. The code for the word is then generated as per the basic Soundex algorithm listed in figure 1 with the length limited to four or six characters for our evaluation in this paper and using the letter mappings given in table 3. The letters c, q and x do not map to a letter code as they are always translated to other letters by transformation rules: c→s or c→k, q→k and x→z or x→ks.

Methodology	Best Match Correct	Best Match Recall	Best Match Retrieved	Best Match Average	Top 10 Correct	Top 10 Recall
Phonetex4	352	0.98	3907	11.1	306	0.85
Phonetex6	340	0.94	1645	4.83	325	0.90
Soundex	330	0.92	6875	20.83	215	0.60
Editex	324	0.90	515	1.43	347	0.96
Agrep	308	0.87	2191	7.02	304	0.84
MS Word 2000	344	0.96				
MS Word 97	339	0.94				
ispell	250	0.69				

Table 5: Table giving the recall and precision for a selection of spell checkers using a test set of 360 phonetic misspellings.

2 Evaluation

For our evaluations we use three dictionaries, each augmented with the correct spelling of our query words where necessary to ensure total coverage of all spelling errors in all lexicons. We use the UNIX 'spell' dictionary containing 29,187 words, the Beale dictionary [4] containing 7,287 words and the CRL dictionary [1] with 45,243 words.

We evaluate our methodology with word code lengths of four and six (Phonetex4 and Phonetex6 respectively) to compare the effect on recall and precision of different code lengths and against the other three methodologies described (Soundex, Agrep and Editex) using 360 phonetic spelling mistakes extracted from various spelling lists [2, 3]. Note that we devised our phonetic code mappings and transformation rules with no prior knowledge of the spelling errors evaluated here. The misspellings range from only one error (insertion, deletion or substitution) from the correct spelling to five errors (for example 'apresteate' for 'appreciate').

For the UNIX 'spell' lexicon in table 5, we list the number of times the correct word is selected with the group of best matches in column 2. The group of best matches may contain a large number of candidate matches so in column 6 we list the number of times the correct word is present in the top 10 matches where the matches are sorted by score and then sorted alphabetically if the scores are equal. This gives two *recall* figures a best match where the number of candidates is unlimited in column 3 and a top 10 only score listed in column 7. For our methodology and Soundex we defined the 'best match' as the set of words matching the highest number of code characters (for example four of four, three of four etc.). We list the total number of words retrieved with all correct best matches in column 4. We then divided this figure by the number of correct best matches to get the average number of words retrieved with a correct best match listed in column 5. The best match average will be lower for methodologies that rank matches (Agrep and Editex) and thus employ finer-grained scoring and grouping rather than those that simply retrieve a single set of best word code matches as with Soundex and our Phonetex. We include the number of correct matches for MS Word 97 & MS Word 2000 and also UNIX 'ispell' as benchmarks. We used the standard supplied dictionaries for both MS Word[1] and 'ispell' so we do not include a

[1]The lexicons included some of the phonetic misspellings as variants of the correct spelling, for example 'miniscule' was stored as a variant of 'minuscule', 'imbed' as a variant of 'embed' in the MS Word lexicon. We counted these as detecting the correct spelling. We also checked that the correct spelling of each word

	Beale		CRL	
Methodology	Best Match Correct	Best Match Recall	Best Match Correct	Best Match Recall
Phonetex4	354	0.98	346	0.96
Soundex	337	0.94	334	0.93
Editex	335	0.93	329	0.91
Agrep	337	0.94	302	0.84

Table 6: Table giving the recall for different lexicon sizes using a test set of 360 phonetic misspellings and the Beale [4] and CRL [1] lexicons.

precision figure for these two benchmarks as we felt a precision comparison would be invalid when the algorithms use different dictionaries.

In table 6 we compare Phonetex4 against Soundex, Editex and Agrep on different lexicon sizes to investigate the effect of lexicon size on best match recall accuracy using the test set of 360 phonetic misspellings. We list the number of times the correct word is selected with the group of best matches in column 2 for the Beale lexicon and column 4 for the CRL lexicon with the corresponding recall figures for Beale and CRL in columns 3 and 5 respectively. We only compare the spell checkers that may be trained with varying size lexicons.

3 Conclusion

From table 5, we can see that our methodology Phonetex4 has the highest recall (best match and top ten inclusive), only failing to find eight words from 360. The best match recall is in fact higher than the recall for Word 2000 for 360 phonetic spelling errors. The average number of words retrieved with the correct best match for Phonetex4 is 11.1, only Soundex has a higher average. However, we intend to integrate the methodology into our existing typographic spell checker [6] so high recall is our motivation as the typographic spelling methods will help minimise the false positives and thus increase precision. Phonetex4 generally fails on phonetic misspellings where a letter has been inserted or deleted. The phonetic word code thus also has a letter code inserted or deleted so the misspelt word code does not align with the word code for the correct spelling. However, typographic spell checkers are designed to overcome such errors, for example n-gram matching [8, 9] so the Phonetex4 errors will be corrected. Zobel and Dart [12] deduced that n-gram error correction substantially outperformed a stand-alone phonetic spell checker for a list of general spelling errors. We feel the phonetic component should be used to augment the existing typographic approach [6] and thus improve recall as [8, 9] identified by handling phonetic errors missed by the typographic approach. [9] identified only a slight improvement but the authors limited the phonetic component to transformation rules applied before the n-gram indexing. We envisage running the phonetic and typographic components in parallel and integrating the candidate matches from each using a suitable fine-grained similarity metric to score and rank the set of candidate matches [6].

We can see from table 5 that Phonetex4 has higher best match recall than Phonetex6.

not correctly matched was present in the lexicon before counting the matches.

Insertion and deletion errors in the phonetic misspellings have more affect on the longer word code and cause fewer correct matches to be retrieved. This agrees with Rogers and Willett's [10] assertion that longer codes increase precision at the expense of recall. If the user required a higher precision than we would recommend our methodology with code length six, which retrieves 4.8 words on average. It has higher best match recall than Agrep, Editex and Soundex and it is certainly not unreasonable for five words to be suggested to the user by a spell checker. Phonetex6 has lower best match retrieval than Editex top 10 retrieval but Editex is slow due to the underlying Edit Distance algorithm which iteratively accumulates a word score with running time $O(mn)$ for a lexicon of m words of average length n. If the user required a stand-alone phonetic spell checker returning the top 10 matches where speed of retrieval is not of overriding importance then Editex would be the optimum methodology of those evaluated but it correctly identifies 5, 19 and 17 fewer words than Phonetex4 best match recall for the UNIX 'spell', Beale and CRL lexicons respectively and is slow due to the $O(mn)$ run time growth.

Table 6 demonstrates that Phonetex maintains high recall across lexicon sizes for best match retrieval ranging from 0.96 for the large 45000+ dictionary to 0.98 for the smaller dictionaries. Phonetex4 outperforms the comparative methods evaluated in table 6 across all lexicon sizes. We feel we have demonstrated the generic use of Phonetex across various lexicon sizes and compositions.

There is a trade-off when developing a phonetic spell checker between including adequate phonetic transformation rules to represent the grammar and maintaining an acceptable retrieval speed. To represent every possible transformation rule would produce an unfeasibly slow system yet sufficient rules need to be included to provide an acceptable quality of retrieval. We feel that our rule base is sufficient for the spell checker we are developing as the phonetic module is intended to integrate with alternative typographic spelling approaches such as those detailed in [7]. We have identified a small rule base of core transformation rules applicable to all English dictionaries to cover general phonetic spelling rules. We have specifically only included rules that apply generally and have few exceptions. We do not account for infrequent exceptions as these would create an intractably large rule base and prevent rapid retrieval; we have mapped sc(e|i|y) $\rightarrow$ s which applies with the exception of sceptic pronounced *skeptic*. However, we feel our recall and coverage are high for the system developed.

4 Acknowledgement

This work was supported by an EPSRC studentship.

References

[1] Web page ftp://ftp.ox.ac.uk/pub/wordlists.

[2] Aspell. Web page http://aspell.sourceforge.net/.

[3] F. Damerau. A technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176, 1964.

[4] Elcom Ltd - Password Recovery Software. Web page http://www.elcomsoft.com/prs.html.

[5] T. Gadd. PHONIX: The Algorithm. *Program*, 24(4):363–366, 1990.

[6] V. Hodge and J. Austin. A Novel Binary Spell Checker. In *Accepted for International Conference on Artificial Neural Networks, Vienna, August*, 2001.

[7] K. Kukich. Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[8] U. Pfeifer, T. Poersch, and N. Fuhr. Retrieval effectiveness of proper name search methods.

[9] A. M. Robertson and P. Willett. Searching for historical word-forms in a database of 17th-century English text using spelling-correction methods. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–265, Copenhagen, Denmark, 1992.

[10] H. J. Rogers and P. Willett. Searching for historical word forms in text databases using spelling-correction methods: Reverse error and phonetic coding methods. *Journal of Documentation*, 47(4):333–353, Dec. 1991.

[11] S. Wu and U. Manber. Fast Text Searching With Errors. *Communications of the ACM*, 35, Oct. 1992.

[12] J. Zobel and P. Dart. Finding approximate matches in large lexicons. *Software Practice and Experience*, 25(3):331–345, Mar. 1995.

[13] J. Zobel and P. Dart. Phonetic String Matching: Lessons from Information Retrieval. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Zurich, Switz, 1996.