IMPROVING PRONUNCIATION DICTIONARY COVERAGE OF NAMES BY MODELLING SPELLING VARIATION

Justin Fackrell and Wojciech Skut

Rhetorical Systems Ltd
4 Crichton's Close
Edinburgh EH8 8DT UK
justin.fackrell@rhetorical.com

ABSTRACT

This paper describes an attempt to improve the coverage of an existing name pronunciation dictionary by modelling variation in spelling. This is done by the derivation of string rewrite rules which operate on out-of-vocabulary words to map them to in-vocabulary words. These string rewrite rules are derived automatically, and are "pronunciation-neutral" in the sense that the mappings they perform on the existing dictionary do not result in a change of pronunciation.

The approach is data-driven, and can be used *online* to make predictions for some (not all) OOV words, or *offline* to add significant numbers of new pronunciations to existing dictionaries. Offline the approach has been used to increase dictionary coverage for four domain-based dictionaries for forenames, surnames, streetnames and placenames. For surnames, a model trained on a 23,000-entry dictionary was subsequently able to add 5,000 new entries, improving both type coverage and token coverage of the dictionaries by about 1%. An informal evaluation suggests that the suggested pronunciations are good in 80% of cases.

1. INTRODUCTION

The pronunciation of out-of-vocabulary (OOV) words is one of the main problems in TTS applications such as automated call centres and car navigation systems. Many of the OOV words are proper names, and these are especially hard to pronounce because they often originate in other languages and they don't behave like other words. The problem is worst for languages like English whose underlying orthography is also highly irregular.

Traditionally this *letter-to-sound* (LTS) problem has been attacked by deriving a set of rules. The rules perform a sequence of substitutions, each one replacing a sequence of graphemes by a (possibly empty) sequence of phonemes.

The actual substitution mechanism can be based on hand-written string replacement rules [1, 2, 3] or it can be learned automatically from data [4, 5]. Unfortunately, the accuracy of such rules is not particularly high, especially on proper names.

In this paper, we describe a novel method for predicting OOV proper names. It is based on a simple but effective principle: mapping an OOV proper name to an in-vocabulary homophone by changing its spelling. The algorithm automatically learns spelling alternations that lead to such homophones in the domain of proper names. The technique doesn't "fire" (i.e. make a prediction) for all OOV names, but when it does, it produces predictions which are phonotactically correct, and it does so without needing grapheme-phoneme alignment (a requirement of some other techniques such as those in [4, 5]).

The paper is organised as follows; we first justify our approach by describing the coverage statistics of the dictionaries we used as the starting point for this work - this illustrates why data-driven techniques are attractive. Then we review hierarchical approaches to LTS, and describe the observations which stimulated the current work. The algorithm is then described in detail, followed by quantitative measures of how the coverage improved, and informal assessment of how good the predictions of the algorithm are. Finally we outline directions in which this work may be developed in future.

1.1. Coverage Requirements

Figure 1 shows how the optimal[1] token coverage and dictionary size are related for four name-and-address domains. The token coverage is calculated using frequency data from an in-house UK postal database of approximately 50 million entries, and the details of each domain sub-database are shown in Table 1. The figure illustrates that small dictionaries of just 1000 entries provide surprisingly large token coverage; the 1000 most common surnames provides over 50% surname token coverage and the 1000 most common forenames provides over 90% forename token cover-

[1] *Optimal* here implies that each dictionary contains those entries which cover the most tokens.

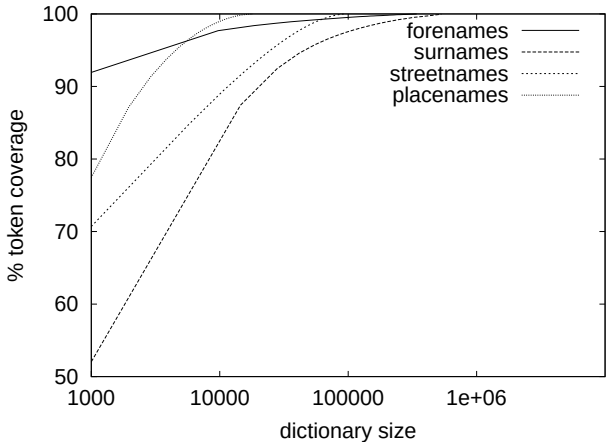

Fig. 1. *Relation between domain-specific dictionary size and optimal token coverage.*

age. However, to attain complete or near-complete token coverage can require very many new types: 100% coverage of surname tokens would require the addition of more than 5,000,000 new entries.

So the number of new dictionary entries that are required to achieve complete coverage is huge, much too large to be added by hand. Automatic methods must therefore be sought which can provide high quality pronunciation predictions for names.

1.2. A Hierarchical Approach

Liberman and Church [6] recognised that the pronunciation dictionary can be viewed as just the first in a series of filters for predicting the pronunciation of a word. In their approach, if a word is not found in the pronunciation dictionary, then attempts to predict the pronunciation are made with a sequence of linguistically-motivated filters – these include the addition of stress-neutral suffixes, rhyming and morphological decomposition. The first filter that fires produces the pronunciation. What all these filters have in common is that they generally do *not* produce output for every input – it is only the last link in the chain which must be able to do that.

With such a hierarchical approach in mind, it makes sense to look for new filters which can make sensible predictions for names which are not in the pronunciation dictionary. A new filter does *not* have to have a very high firing rate. All that is required for it to be useful is that, when it does fire, it produces predictions of a higher accuracy than the links in the chain below it. From literature the quality of predictions of automatically trained pronunciation rules is in the region of 70-75% [4, 7] and the best results of other techniques seem to be lower [5]. Therefore any filters with a higher success rate than this have potential for improving the quality of the system. In this paper we propose an automatically trained filter which has a modest firing rate, but which produces predictions which are judged to be good approximately 80% of the time.

1.3. LTS is a many-to-one Mapping

The current work was motivated by the observation that, within a medium-sized surnames dictionary for RP English, roughly 10% of ways of pronouncing a name have more than one spelling. This is illustrated in Table 1 which shows, for each domain dictionary, the numbers of unique orthographic and phonetic entries.

Table 1. *Characteristics of pronunciation dictionaries used in this paper. N_{orth} is the number of dictionary entries (headwords), N_{pron} is the number of distinct pronunciations, and N_{multi} is the number (percentage in brackets) of pronunciations which have more than one spelling.*

	N_{orth}	N_{pron}	N_{multi} (%)
forenames	14962	13479	1747 (13.0)
surnames	23746	21487	2641 (12.3)
streetnames	16211	15267	1358 (8.9)
placenames	3668	3680	153 (4.2)

Thus given a list of names which are not in a particular dictionary, we hypothesize that about 10% of these names *do* already have a valid pronunciation in the dictionary. The LTS problem for these names is then the task of finding the mapping from OOV to in-vocabulary. In other words the task is to try to find a homophone entry in the existing dictionary.

This problem is closely related to one in the field of "name retrieval", in which database queries are made more useful by allowing fuzziness in name matching. In name retrieval, the nearest matches to a search key (i.e. a name) are returned as "hits". These hits are found using a variety of methods (reviewed in [8, 9]) which typically involve the calculation of a distance between the key and each name in the database.

The oldest of these techniques, Soundex and Phonix, perform the distance measure implicitly by attempting to map each word to a representation shared by its "soundalikes". Soundex correctly identifies the names "Reynold" and "Reynauld" as soundalikes, but it also pairs "Catherine" and "Cotroneo"[8]. Explicit string edit distances have also been used in name retrieval, primarily for the identification of typing errors [9].

Further developments have seen the combination of explicit string edit distances with phonetically-motivated substring transformations. The link with phonetics was made explicit in Zobel and Dart's [8] phonometric approach: LTS

rules are used to predict pronunciations of search keys, and the distance metric is calculated in the phonetic domain. While this may provide some improvement for name retrieval systems, the reliance on LTS rules is an obvious weakness in the approach, and the examples provided in [8] suggest that soundalikes identified by this method are phonetically diverse i.e. that they are rarely homophones.

If a name retrieval technique could be found which only identified homophone matches, then this could be used to find pronunciations of OOV words by identify their in-vocabulary soundalikes. This is the goal of the current work.

2. THE ALGORITHM

The current work is based on the idea that, within a particular domain (e.g. surnames), there exist universal spelling alternations which are pronunciation-neutral. That is, there are ways in which the spelling of a word can be changed without changing its pronunciation.

The variation in spelling can be modelled by finding string rewrite rules which are pronunciation-neutral in an existing pronunciation dictionary. Given an OOV name, the algorithm tries to find a string rewrite rule which rewrites the name to an in-vocabulary spelling. If it succeeds, then it has found a homophone for the OOV word, and the pronunciation can simply be looked up in the dictionary.

The algorithm will now be described in detail, first by showing how the model for spelling variation is trained from an existing dictionary, and then by discussing how the model is used to make pronunciation predictions for words which are OOV.

2.1. Training

The starting point for training is a dictionary which gives partial coverage of the domain in question. We favour using a domain-specific dictionary for this rather than a general purpose dictionary, since we suspect that the nature of spelling variation is domain-dependent.

The first stage is to create a reverse dictionary, which maps pronunciations to orthography. All entries in the reverse dictionary which map one pronunciation to just one spelling are then removed. For the remainder, each pair of spellings which share a pronunciation are used to generate a sequence of rewrite rules $r_i, i = 0, 1, .., n-1$. Each rewrite rule r_i is of the form `A` → `B / L _ R` where the pattern `A` with `L` as left context and `R` as right context is replaced with the string `B`.

Consider an example: the pronunciation `/ l i1 n . z ii2 /` is shared by the spellings **linsey** and **lynsey** (**linsey**=**lynsey**). Table 2 shows the postulated rewrite rules.

The first rewrite rule is obtained by identifying, then removing, the common prefix and suffix between the two strings, to yield a simple context-free substitution string (e.g. `i → y / _`). The second and subsequent rules are obtained by successively adding extra context information, first to the right, then to the left, where possible.

Table 2. *Rules r_i postulated from the soundalike pair* **linsey**=**lynsey**.

i	substitution rule r_i
0	`i → y /  _`
1	`i → y /  _ n`
2	`i → y / ^ l _ n`
3	`i → y / ^ l _ ns`
4	`i → y / ^ l _ nse`
5	`i → y / ^ l _ nsey$`

The rules at the top of the list will fire most often, but will frequently map names to other names with *different* pronunciations (e.g. **smith**≠**smyth**). Conversely, the rule at the bottom of the list will fire only once mapping the original word pair **linsey**=**lynsey**.

Each of the rules r_i is evaluated on the rest of the dictionary. For each entry in the dictionary, a particular rule will do one of four things:

MISS The pattern doesn't match (e.g. **bilton**[2])

OOV The pattern matches, but the resulting mapping is not in the dictionary (e.g. **linton**→**lynton**, but **lynton** is OOV)

DIFF The pattern matches, the resulting mapping *is* in the dictionary, but the pronunciations are different (e.g. **tin** → **tyn**, but `/t i1 n/` ≠ `/t ii1 n/`

GOOD The pattern matches, the resulting mapping *is* in the dictionary, *and* the pronunciations are the same. (e.g. **linne** → **lynne**, and both are pronounced `/l i1 n/`)

Counting over the whole dictionary, each rule r_i is assigned four scores: n_i^{MISS}, n_i^{OOV}, n_i^{DIFF} and n_i^{GOOD}. Collectively these scores reflect how useful the rule is – how often it can be expected to fire, how often it will map into the dictionary, and how often it makes a pronunciation-neutral mapping.

Of the r_i, just one rule is chosen for inclusion in the rule set. Currently, the heuristic for choosing the best rule from each set is simply to choose the shortest rule which is *always* pronunciation-neutral when its pattern matches and it maps into the dictionary ($n_i^{DIFF} = 0$). In future it may be advantagous to add sophistication to this part of the technique.

The above process is repeated for all other spelling pairs, to yield a list of substitution rules.

[2] All examples in this list apply to rule r_1 in Table 2.

2.2. Prediction

The substitution rules are scored and then sorted by their relevance – which is simply the count of how many successful mappings they make in the existing dictionary. For any OOV word, we find the highest-scoring substitution rule which maps the OOV word into the dictionary, and then use the pronunciation of that word.

This can be done offline to generate new dictionary entries, or live at synthesis time. In the current work, prediction is done offline, generating phonetic transcriptions for a given list of words that are not in the available pronunciation dictionary. The transcriptions are then added to the pronunciation dictionary.

Two objections can be made to this approach:

1. The offline approach restricts the coverage of the new lookup method to a predefined set of OOV words although lookup at synthesis time would enable the system to map unseen OOV words to existing pronunciations. However, the application under consideration (UK proper names) means that the domain – although very large – is practically finite and can be covered by a list of words. Furthermore, the approach taken is not guaranteed to perform equally well on material different from proper names.

2. Putting the missing words into the dictionary may be costly in terms of memory. However, memory is generally cheap and the use of efficient representations such as finite-state machines [10, 11] can mean that this cost is in fact moderate. In the implementation reported in the present paper, a pronunciation dictionary containing over 440K entries was encoded as a finite-state transducer and then minimised, yielding a finite-state transducer with 215,540 states and 549,538 transitions, using less than 8MB of RAM. This figure can be reduced even further by means of automata compression [12].

3. EVALUATION

To evaluate the technique, a set of base dictionaries were used which provide basic coverage of four domains – *forenames*, *surnames*, *streetnames* and *placenames*.

The algorithm was used to derive rewrite rules on each of the four domains of interest, resulting in four sets of rewrite rules. The size of these rule sets, plus some example rules, are shown in Table 3.

These rewrite rule sets were then used to make predictions for the remaining OOV words for each domain.

Table 4 shows the percentage improvement in coverage for the dictionaries obtained by using the algorithm. Clearly the change in coverage is only a small improvement, but bear in mind that since the number of types and tokens in the population is very large, this small improvement does in fact represent several thousand new dictionary entries. (As far as token coverage is concerned, a 1% improvement in UK surname coverage means that about half a million people will find their name in the dictionary)

Table 3. *Rewrite rules trained from base dictionaries. n_{new} is the number of previously OOV spellings added as a result of the rule.*

	no. of rules	highest scoring rules	n_{new}
forenames	667	a → / _ a	126
		y → i / _ l	99
		gh → / a _ $	63
		igh → y / _ $	59
surnames	1081	y → i / _ l	94
		ey → ai / _	64
		n → / o _ n$	60
		all → le / _ $	56
streetnames	702	igh → y / _ $	57
		' → / _	42
		s → 's / _ $	32
		y → i / _ l	31
placenames	49	t → / _ t$	3
		e → / k _ $	3
		n → / n _ $	3
		t → et / _	2

Table 4. *Coverage of dictionary (in %) before and after application of spelling variation algorithm on the pronunciation dictionaries described in Table 1 (FN=forenames, SN=surnames, ST=streetnames, PL=placenames).*

	type			token		
dom.	before	after	Δ	before	after	Δ
FN	4.3	5.3	+1.0	94.9	95.3	+0.4
SN	4.4	5.3	+0.9	75.2	76.2	+1.0
ST	16.9	18.1	+1.2	81.6	82.0	+0.4
PL	19.2	19.4	+0.2	75.2	75.2	0.0

Further experiments with larger dictionaries suggest that the algorithm remains effective at mapping OOV words into the dictionary even when token coverage is 98% and higher.

To see whether the mappings suggested by the rewrite rule algorithm are actually any good, an evaluation experiment was carried out. For each domain, a random test set was constructed consisting of OOV names for which the respelling algorithm had found new spellings. For placenames, the algorithm only identified 37 new spellings, so

all of these were used in the test. For the other domains, 200 names were used.

Each stimulus consists of a pair of words: an OOV name and the in-vocabulary soundalike identified by the algorithm (e.g. **donelly→donelley**). Subjects were shown the spellings of both words, and asked to rate each soundalike with the value 1 ("these two words are pronounced the same") or 0 ("these two words are not pronounced the same", or "I don't know"). Within each domain, the same pairs were shown to each subject. The experiment was carried out by five native British English speakers.

Table 5 shows the results from the listening test. The predictions of the rewrite algorithm are good, with average scores between 80% and 90%. Even if unanimity between all 5 judges is required (N_5 in the table), the results remain encouraging.

Table 5. *Subjective evaluation of rewrite rules. n is the number of test words. S is the percentage of "good" words. N_5 is the percentage of test words which were judged "good" by all 5 subjects.*

domain	n	S %	N_5 %
forenames	200	90.2	70.5
surnames	200	80.7	61.5
streetnames	200	88.6	69.5
placenames	37	85.2	81.5

Table 6 shows examples of successes of the algorithm, in which the mapping was judged "good". The transformations which occur are undoubtedly simple, and may well be produced by other rule-based approaches such as that of [6]. However, the transformation rules presented here were inferred fully automatically from an existing dictionary, and so the applicability of the technique to other domains and languages appears possible.

Table 7 shows examples of failures of the algorithm – what remains to be investigated is what the correlation is between the rule relevance (i.e. how much evidence for the rule is there in the current dictionary) and the quality of the predictions it makes.

4. CONCLUSIONS AND FURTHER WORK

In this paper an algorithm has been proposed which contributes to lexical coverage for names by finding in-vocabulary spelling variants for OOV words. The resulting rule sets do not fire with a high frequency, but in an experiment based on a UK database, are able to improve token coverage by approximately 1%, which corresponds to about half a million people. An informal evaluation suggests that for those OOV words for which the algorithm does suggest pronunciations,

Table 6. *Examples of rewrites judged "good".*

forenames	hailee → hailey kymberleigh → kymberley mycheala→ micheala
surnames	whatkinson → watkinson geoffreys → jeffreys casy → casey
streetnames	strangways → strangeways ailesbury → aylesbury macks → max
placenames	whelford → welford holmer → homer lorton → laughton

Table 7. *Examples of rewrites judged "bad".*

forenames	cansey → kasey charistos→ christos jitendera→ jitendra
surnames	nelon→ nelsen shazde → shazad moli → morley
streetnames	beechers → beeches bedes → beds cloch → clough
placenames	ston → seton prehen → preen longswood→ longwood

about 80% are good, with a high degree of agreement between the subjects.

One useful property of the technique is that all the predictions it produces are phonotactically correct, since it is mapping new words into the existing dictionary. Some rule based methods such as CART are not constrained in such a way.

It is hoped that this approach can form part of a battery of letter-to-sound approaches to improve dictionary coverage of names.

The algorithm in its current form is fairly simple, and there is no capacity for more than one rewrite rule to fire on a particular OOV name. This is something which will be investigated in future.

Further experiments are warranted to investigate the behaviour of the algorithm on larger dictionaries, when token coverage is approaching 100%, and work is also required to add sophistication to the context rules.

Finally, the online applicability of the method described in this paper presents a promising research prospect. If, in addition to proper names, the algorithm turns out to perform well on arbitrary input data, applying the rewrite mechanism

at synthesis time will increase the coverage of the method beyond the predefined list of OOV words. For this, an efficient lookup method is needed that would find the best applicable mapping deterministically for a given string. The finite-state framework used to encode the pronunciation dictionary in our system offers several efficient methods for performing this kind of lookup [13, 14].

5. REFERENCES

[1] Honey S. Elovitz, Rodney Johnson, Astrid McHugh, and John E. Shore, "Letter-to-sound rules for automatic translation of english text to phonetics," in *IEEE Transactions on Acoustics, Speech and Signal Processing ASSP-24*, 1976, pp. 446–459.

[2] Mehryar Mohri and Richard Sproat, "An efficient compiler for weighted rewrite rules," in *Meeting of the Association for Computational Linguistics*, 1996, pp. 231–238.

[3] I. Lee Hetherington, "An efficient implementation of phonological rules using finite-state transducers," in *Proceedings of Eurospeech 2001*, 2001.

[4] A. Black, K. Lenzo, and V. Pagel, "Issues in building general letter to sound rules," in *Proceedings of ESCA/COCOSDA Workshop on Speech Synthesis*, Jenolan Caves, Australia, 1998.

[5] Yannick Marchand and Robert I. Damper, "A multistrategy approach to improving pronunciation by analogy," *Computational Linguistics*, vol. 26, no. 2, pp. 195–219, 2000.

[6] M. Liberman and K. Church, "Text analysis and word pronunciation in text-to-speech synthesis," in *Advances in Speech Signal Processing*, S. Furui and M. Sondhi, Eds. Marcel Dekker Inc, 1991.

[7] Ariadna Font Llitjos and Alan W Black, "Knowledge of language origin improves pronunciation accuracy of proper names," 2001.

[8] J. Zobel and P. W. Dart, "Phonetic string matching: Lessons from information retrieval," in *Proceedings of the 19th International Conference on Research and Development in Information Retrieval*, H.-P. Frei, D. Harman, P. Schäble, and R. Wilkinson, Eds., Zurich, Switzerland, 1996, pp. 166–172, ACM Press.

[9] Ulrich Pfeifer, Thomas Poersch, and Norbert Fuhr, "Searching proper names in databases," in *Hypertext - Information Retrieval - Multimedia*, pp. 259–276. Universitätsverlag Konstanz, 1995.

[10] Mehryar Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, 1997.

[11] Stoyan Mihov and Denis Maurel, "Direct construction of minimal acyclic subsequential transducers," *Lecture Notes in Computer Science*, vol. 2088, 2001.

[12] Jan Daciuk, "Experiments with automata compression," *Lecture Notes in Computer Science*, vol. 2088, pp. 105–112, 2001.

[13] Kemal Oflazer, "Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction," *Computational Linguistics*, vol. 22, no. 1, pp. 73–89, 1996.

[14] M. Crochemore and C. Hancart, "Automata for matching patterns," in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds., vol. 2, pp. 399–462. Springer-Verlag, 1997.