Applying Many-to-Many Alignments and Hidden Markov Models to Letter-to-Phoneme Conversion

Sittichai Jiampojamarn, Grzegorz Kondrak and Tarek Sherif
Department of Computing Science,
University of Alberta,
Edmonton, AB, T6G 2E8, Canada
{sj,kondrak,tarek}@cs.ualberta.ca

Abstract

Letter-to-phoneme conversion generally requires aligned training data of letters and phonemes. Typically, the alignments are limited to one-to-one alignments. We present a novel technique of training with many-to-many alignments. A letter chunking bigram prediction manages double letters and double phonemes automatically as opposed to preprocessing with fixed lists. We also apply an HMM method in conjunction with a local classification model to predict a global phoneme sequence given a word. The many-to-many alignments result in significant improvements over the traditional one-to-one approach. Our system achieves state-of-the-art performance on several languages and data sets.

1 Introduction

Letter-to-phoneme (L2P) conversion requires a system to produce phonemes that correspond to a given written word. Phonemes are abstract representations of how words should be pronounced in natural speech, while letters or graphemes are representations of words in written language. For example, the phonemes for the word *phoenix* are [f i n ɪ k s].

The L2P task is a crucial part of speech synthesis systems, as converting input text (graphemes) into phonemes is the first step in representing sounds. L2P conversion can also help improve performance in spelling correction (Toutanova and Moore, 2001). Unfortunately, proper nouns and unseen words prevent a table look-up approach. It is infeasible to construct a lexical database that includes every word in the written language. Likewise, orthographic complexity of many languages prevents us from using hand-designed conversion rules. There are always exceptional rules that need to be added to cover a large vocabulary set. Thus, an automatic L2P system is desirable.

Many data-driven techniques have been proposed for letter-to-phoneme conversion systems, including pronunciation by analogy (Marchand and Damper, 2000), constraint satisfaction (Van Den Bosch and Canisius, 2006), Hidden Markov Model (Taylor, 2005), decision trees (Black et al., 1998), and neural networks (Sejnowski and Rosenberg, 1987). The training data usually consists of written words and their corresponding phonemes, which are not aligned; there is no explicit information indicating individual letter and phoneme relationships. These relationships must be postulated before a prediction model can be trained.

Previous work has generally assumed one-to-one alignment for simplicity (Daelemans and Bosch, 1997; Black et al., 1998; Damper et al., 2005). An expectation maximization (EM) based algorithm (Dempster et al., 1977) is applied to train the aligners. However, there are several problems with this approach. Letter strings and phoneme strings are not typically the same length, so null phonemes and null letters must be introduced to make one-to-one-alignments possible, Furthermore, two letters frequently combine to produce a single phoneme (double letters), and a single letter can sometimes produce two phonemes (double phonemes).

To help address these problems, we propose an automatic many-to-many aligner and incorporate it into a generic classification predictor for letter-to-phoneme conversion. Our many-to-many aligner automatically discovers double phonemes and double letters, as opposed to manually preprocessing data by merging phonemes using fixed lists. To our knowledge, applying many-to-many alignments to letter-to-phoneme conversion is novel.

Once we have our many-to-many alignments, we use that data to train a prediction model. Many phoneme prediction systems are based on local prediction methods, which focus on predicting an individual phoneme given each letter in a word. Conversely, a method like pronunciation by analogy (PbA) (Marchand and Damper, 2000) is considered a global prediction method: predicted phoneme sequences are considered as a whole. Recently, Van Den Bosch and Canisius (2006) proposed trigram class prediction, which incorporates a constraint satisfaction method to produce a global prediction for letter-to-phoneme conversion. Both PbA and trigram class prediction show improvement over predicting individual phonemes, confirming that L2P systems can benefit from incorporating the relationship between phonemes in a sequence.

In order to capitalize on the information found in phoneme sequences, we propose to apply an HMM method after a local phoneme prediction process. Given a candidate list of two or more possible phonemes, as produced by the local predictor, the HMM will find the best phoneme sequence. Using this approach, our system demonstrates an improvement on several language data sets.

The rest of the paper is structured as follows. We describe the letter-phoneme alignment methods including a standard one-to-one alignment method and our many-to-many approach in Section 2. The alignment methods are used to align graphemes and phonemes before the phoneme prediction models can be trained from the training examples. In Section 3, we present a letter chunk prediction method that automatically discovers double letters in grapheme sequences. It incorporates our many-to-many alignments with prediction models. In Section 4, we present our application of an HMM method to the local prediction results. The results of experiments on several language data sets are discussed in Section 5. We conclude and propose future work in Section 6.

2 Letter-phoneme alignment

2.1 One-to-one alignment

There are two main problems with one-to-one alignments:

1. Double letters: two letters map to one phoneme (e.g. *sh* - [ʃ], *ph* - [f]).

2. Double phonemes: one letter maps to two phonemes (e.g. *x* - [k s], *u* - [j u]).

First, consider the double letter problem. In most cases when the grapheme sequence is longer than the phoneme sequence, it is because some letters are silent. For example, in the word *abode*, pronounced [ə b o d], the letter e produces a null phoneme (ϵ). This is well captured by one-to-one aligners. However, the longer grapheme sequence can also be generated by double letters; for example, in the word *king*, pronounced [k ɪ ŋ], the letters *ng* together produce the phoneme [ŋ]. In this case, one-to-one aligners using null phonemes will produce an incorrect alignment. This can cause problems for the phoneme prediction model by training it to produce a null phoneme from either of the letters n or g.

In the double phoneme case, a new phoneme is introduced to represent a combination of two (or more) phonemes. For example, in the word *fume* with phoneme sequence [f j u m], the letter u produces both the [j] and [u] phonemes. There are two possible solutions for constructing a one-to-one alignment in this case. The first is to create a new phoneme by merging the phonemes [j] and [u]. This requires constructing a fixed list of new phonemes before beginning the alignment process. The second solution is to add a null letter in the grapheme sequence. However, the null letter not only confuses the phoneme prediction model, but also complicates the the phoneme generation phase.

For comparison with our many-to-many approach, we implement a one-to-one aligner based on the epsilon scattering method (Black et al., 1998). The method applies the EM algorithm to estimate

Algorithm 1: Pseudocode for a many-to-many expectation-maximization algorithm.

Algorithm:EM-many2many
Input: $x^T, y^V, maxX, maxY$
Output: γ

forall *mapping operations* z **do**
    $\gamma(z) := 0$
foreach *sequence pair* (x^T, y^V) **do**
    Expectation-many2many$(x^T, y^V, maxX, maxY, \gamma)$
Maximization-Step(γ)

Algorithm 2: Pseudocode for a many-to-many expectation algorithm.

Algorithm: Expectation-many2many
Input: $x^T, y^V, maxX, maxY, \gamma$
Output: γ

$\alpha :=$ *Forward-many2many* $(x^T, y^V, maxX, maxY)$
$\beta :=$ *Backward-many2many* $(x^T, y^V, maxX, maxY)$
if $(\alpha_{T,V} = 0)$ **then**
    return
for $t = 0...T$ **do**
    for $v = 0...V$ **do**
        if $(t > 0 \wedge DELX)$ **then**
            for $i = 1...maxX$ **st** $t - i \geq 0$ **do**
                $\gamma(x^t_{t-i+1}, \epsilon) += \frac{\alpha_{t-i,v}\delta(x^t_{t-i+1},\epsilon)\beta_{t,v}}{\alpha_{T,V}}$
        if $(v > 0 \wedge DELY)$ **then**
            for $j = 1...maxY$ **st** $v - j \geq 0$ **do**
                $\gamma(\epsilon, y^v_{v-j+1}) += \frac{\alpha_{t,v-j}\delta(\epsilon,y^v_{v-j+1})\beta_{t,v}}{\alpha_{T,V}}$
        if $(v > 0 \wedge t > 0)$ **then**
            for $i = 1...maxX$ **st** $t - i \geq 0$ **do**
                for $j = 1...maxY$ **st** $v - j \geq 0$ **do**
                    $\gamma(x^t_{t-i+1}, y^v_{v-j+1}) += \frac{\alpha_{t-i,v-j}\delta(x^t_{t-i+1},y^v_{v-j+1})\beta_{t,v}}{\alpha_{T,V}}$

the probability of mapping a letter l to a phoneme p, $P(l, p)$. The initial probability table starts by mapping all possible alignments between letters and phonemes for each word in the training data, introducing all possible null phoneme positions. For example, the word/phoneme-sequence pair *abode* [ə b o d] has five possible positions where a null phoneme can be added to make an alignment.

The training process uses the initial probability table $P(l, p)$ to find the best possible alignments for each word using the Dynamic Time Warping (DTW) algorithm (Sankoff and Kruskal, 1999). At each iteration, the probability table $P(l, p)$ is re-calculated based on the best alignments found in that iteration. Finding the best alignments and re-calculating the probability table continues iteratively until there is no change in the probability table. The final probability table $P(l, p)$ is used to find one-to-one alignments given graphemes and phonemes.

2.2 Many-to-Many alignment

We present a many-to-many alignment algorithm that overcomes the limitations of one-to-one aligners. The training of the many-to-many aligner is an extension of the forward-backward training of a one-to-one stochastic transducer presented in (Ristad and Yianilos, 1998). Partial counts are counts of all possible mappings from letters to phonemes that are collected in the γ table, while mapping probabilities (initially uniform) are maintained in the δ table. For each grapheme-/phoneme-sequence pair (x, y), the *EM-many2many* function (Algorithm 1) calls the *Expectation-many2many* function (Algorithm 2) to collect partial counts. T and V are the lengths of x and y respectively. The $maxX$ and $maxY$ variables are the maximum lengths of subsequences used in a single mapping operation for x and y. (For the task at hand, we set both $maxX$ and $maxY$ to 2.) The *Maximization-step* function simply normalizes the partial counts to create a probability distribution. Normalization can be done over the whole table to create a joint distribution or per grapheme to create a conditional distribution.

The *Forward-many2many* function (Algorithm 3) fills in the table α, with each entry $\alpha(t, v)$ being the sum of all paths through the transducer that generate the sequence pair (x^t_1, y^v_1). Analogously, the *Backward-many2many* function fills in β, with each entry $\beta(t, v)$ being the sum of all paths through the transducer that generate the sequence pair (x^T_t, y^V_v). The constants $DELX$ and $DELY$ indicate whether or not deletions are allowed on either side. In our system, we allow letter deletions (i.e. mapping of letters to null phoneme), but not phoneme deletions.

Expectation-many2many first calls the two functions to fill the α and β tables, and then uses the probabilities to calculate partial counts for every possible mapping in the sequence pair. The partial count collected at positions t and v in the sequence pair is the sum of all paths that generate the sequence pair and go through (t, v), divided by the sum of all paths that generate the entire sequence pair ($\alpha(T, V)$).

Once the probabilities are learned, the Viterbi

Algorithm 3: Pseudocode for a many-to-many forward algorithm.

```
Algorithm:Forward-many2many
Input: (x^T, y^V, maxX, maxY)
Output: α

α_{0,0} := 1
for t = 0...T do
  for v = 0...V do
    if (t > 0 ∨ v > 0) then
      α_{t,v} = 0
    if (t > 0 ∧ DELX) then
      for i = 1...maxX st t − i ≥ 0 do
        α_{t,v} += δ(x^t_{t−i+1}, ε) α_{t−i,v}
    if (v > 0 ∧ DELY) then
      for j = 1...maxY st v − j ≥ 0 do
        α_{t,v} += δ(ε, y^v_{v−j+1}) α_{t,v−j}
    if (v > 0 ∧ t > 0) then
      for i = 1...maxX st t − i ≥ 0 do
        for j = 1...maxY st v − j ≥ 0 do
          α_{t,v} += δ(x^t_{t−i+1}, y^v_{v−j+1}) α_{t−i,v−j}
```

algorithm can be used to produce the most likely alignment as in the following equations. Back pointers to maximizing arguments are kept at each step so the alignment can be reconstructed.

$$\alpha(0,0) = 1 \tag{1}$$

$$\alpha(t,v) = \max_{\substack{1\leq i \leq maxX,\\ 1\leq j\leq maxY}} \begin{cases} \delta(x^t_{t-i+1},\epsilon)\alpha_{t-i,v} \\ \delta(\epsilon, y^v_{v-j+1})\alpha_{t,v-j} \\ \delta(x^t_{t-i+1}, y^v_{v-j+1})\alpha_{t-i,v-j} \end{cases} \tag{2}$$

Given a set of words and their phonemes, alignments are made across graphemes and phonemes. For example, the word *phoenix*, with phonemes [f i n ɪ k s], is aligned as:

ph	*oe*	*n*	*i*	*x*
|	|	|	|	|
f	i	n	ɪ	ks

The letters *ph* are an example of the double letter problem (mapping to the single phoneme [f]), while the letter x is an example of the double phoneme problem (mapping to both [k] and [s] in the phoneme sequence). These alignments provide more accurate grapheme-to-phoneme relationships for a phoneme prediction model.

3 Letter chunking

Our new alignment scheme provides more accurate alignments, but it is also more complex — sometimes a prediction model should predict two phonemes for a single letter, while at other times the prediction model should make a prediction based on a pair of letters. In order to distinguish between these two cases, we propose a method called "letter chunking".

Once many-to-many alignments are built across graphemes and phonemes, each word contains a set of letter chunks, each consisting of one or two letters aligned with phonemes. Each letter chunk can be considered as a grapheme unit that contains either one or two letters. In the same way, each phoneme chunk can be considered as a phoneme unit consisting of one or two phonemes. Note that the double letters and double phonemes are implicitly discovered by the alignments of graphemes and phonemes. They are not necessarily consistent over the training data but based on the alignments found in each word.

In the phoneme generation phase, the system has only graphemes available to predict phonemes, so there is no information about letter chunk boundaries. We cannot simply merge any two letters that have appeared as a letter chunk in the training data. For example, although the letter pair *sh* is usually pronounced as a single phoneme in English (e.g. *gash* [g ae ʃ]), this is not true universally (e.g. *gasholder* [g ae s h o l d ə r]). Therefore, we implement a letter chunk prediction model to provide chunk boundaries given only graphemes.

In our system, a bigram letter chunking prediction automatically discovers double letters based on instance-based learning (Aha et al., 1991). Since the many-to-many alignments are drawn from 1-0, 1-1, 1-2, 2-0, and 2-1 relationships, each letter in a word can form a chunk with its neighbor or stand alone as a chunk itself. We treat the chunk prediction as a binary classification problem. We generate all the bigrams in a word and determine whether each bigram should be a chunk based on its context. Table 1 shows an example of how chunking prediction proceeds for the word *longs*. Letters $l_{i-2}, l_{i-1}, l_{i+1}$, and l_{i+2} are the context of the bigram l_i; $chunk = 1$ if the letter bigram l_i is a chunk. Otherwise, the chunk simply consists of an individual letter. In the example, the word is decomposed as $l|o|ng|s$, which can be aligned with its pronunciation [l | ɒ | ŋ | z]. If the model happens to predict consecutive overlapping chunks, only the first of the two is accepted.

l_{i-2}	l_{i-1}	l_i	l_{i+1}	l_{i+2}	$chunk$
_	_	lo	n	g	0
_	l	on	g	s	0
l	o	ng	s	_	1
o	n	gs	_	_	0

Table 1: An example of letter chunking prediction.

4 Phoneme prediction

Most of the previously proposed techniques for phoneme prediction require training data to be aligned in one-to-one alignments. Those models approach the phoneme prediction task as a classification problem: a phoneme is predicted for each letter independently without using other predictions from the same word. These local predictions assume independence of predictions, even though there are clearly interdependencies between predictions. Predicting each phoneme in a word without considering other assignments may not satisfy the main goal of finding a set of phonemes that work together to form a word.

A trigram phoneme prediction with constraint satisfaction inference (Van Den Bosch and Canisius, 2006) was proposed to improve on local predictions. From each letter unit, it predicts a trigram class that has the target phoneme in the middle surrounded by its neighboring phonemes. The phoneme sequence is generated in such a way that it satisfies the trigram, bigram and unigram constraints. The overlapping predictions improve letter-to-phoneme performance mainly by repairing imperfect one-to-one alignments.

However, the trigram class prediction tends to be more complex as it increases the number of target classes. For English, there are only 58 unigram phoneme classes but 13,005 tri-gram phoneme classes. The phoneme combinations in the tri-gram classes are potentially confusing to the prediction model because the model has more target classes in its search space while it has access to the same number of local features in the grapheme side.

We propose to apply a supervised HMM method embedded with local classification to find the most likely sequence of phonemes given a word. An HMM is a statistical model that combines the observation likelihood (probability of phonemes given letters) and transition likelihood (probability of current phoneme given previous phonemes) to predict each phoneme. Our approach differs from a basic Hidden Markov Model for letter-to-phoneme system (Taylor, 2005) that formulates grapheme sequences as observation states and phonemes as hidden states. The basic HMM system for L2P does not provide good performance on the task because it lacks context information on the grapheme side. In fact, a pronunciation depends more on graphemes than on the neighboring phonemes; therefore, the transition probability (language model) should affect the prediction decisions only when there is more than one possible phoneme that can be assigned to a letter.

Our approach is to use an instance-based learning technique as a local predictor to generate a set of phoneme candidates for each letter chunk, given its context in a word. The local predictor produces confidence values for Each candidate phoneme. We normalize the confidence values into values between 0 and 1, and treat them as the emission probabilities, while the transition probabilities are derived directly from the phoneme sequences in the training data.

The pronunciation is generated by considering both phoneme prediction values and transition probabilities. The optimal phoneme sequence is found with the Viterbi search algorithm. We limit the size of the context to $n = 3$ in order to avoid overfitting and minimize the complexity of the model. Since the candidate set is from the classifier, the search space is limited to a small number of candidate phonemes (1 to 5 phonemes in most cases).

The HMM postprocessing is independent of local predictions from the classifier. Instead, it selects the best phoneme sequence from a set of possible local predictions by taking advantage of the phoneme language model, which is trained on the phoneme sequences in the training data.

5 Evaluation

We evaluated our approaches on CMUDict, Brulex, and German, Dutch and English Celex corpora (Baayen et al., 1996). The corpora (except English Celex) are available as part of the Letter-to-Phoneme Conversion PRONALSYL Challenge[1].

[1]The PRONALSYL Challenge: `http://www.pascal-network.org/Challenges/PRONALSYL/`.

Language	Data set	Number of words
English	CMUDict	112,102
English	Celex	65,936
Dutch	Celex	116,252
German	Celex	49,421
French	Brulex	27,473

Table 2: Number of words in each data set.

For the English Celex data, we removed duplicate words as well as words shorter than four letters. Table 2 shows the number of words and the language of each corpus.

For all of our experiments, our local classifier for predicting phonemes is the instance-based learning IB1 algorithm (Aha et al., 1991) implemented in the TiMBL package (Daelemans et al., 2004). The HMM technique is applied as post processing to the instance-based learning to provide a sequence prediction. In addition to comparing one-to-one and many-to-many alignments, we also compare our method to the constraint satisfaction inference method as described in Section 4. The results are reported in word accuracy rate based on the 10-fold cross validation, with the mean and standard deviation values.

Table 3 shows word accuracy performance across a variety of methods. We show results comparing the one-to-one aligner described in Section 2.1 and the one-to-one aligner provided by the PRONALSYL challenge. The PRONALSYS one-to-one alignments are taken directly from the PRONALSYL challenge, whose method is based on an EM algorithm. For both alignments, we use instance-based learning as the prediction model.

Overall, our one-to-one alignments outperform the alignments provided by the data sets for all corpora. The main difference between the PRONALSYS one-to-one alignment and our one-to-one alignment is that our aligner does not allow a null letter on the grapheme side. Consider the word *abomination* [ə b ɒ m ɪ n e ʃ ə n]: the first six letters and phonemes are aligned the same way by both aligners (*abomin-* [ə b ɒ m ɪ n]). However, the two aligners produce radically different alignments for the last five letters. The alignment provided by the PRONALSYS one-to-one alignments is:

_	_	*a*	*t*	*i*	*o*	*n*
|	|	|	|	|	|	|
e	ʃ	ə	_	_	_	n

while our one-to-one alignment is:

a	*t*	*i*	*o*	*n*
|	|	|	|	|
e	_	ʃ	ə	n

Clearly, the latter alignment provides more information on how the graphemes map to the phonemes.

Table 3 also shows that impressive improvements for all evaluated corpora are achieved by using many-to-many alignments rather than one-to-one alignments (1-1 align vs. M-M align). The significant improvements, ranging from 2.7% to 7.6% in word accuracy, illustrate the importance of having more precise alignments. For example, we can now obtain the correct alignment for the second part of the word *abomination*:

a	*ti*	*o*	*n*
|	|	|	|
e	ʃ	ə	n

Instead of adding a null phoneme in the phoneme sequence, the many-to-many aligner maps the letter chunk *ti* to a single phoneme.

The HMM approach is based on the same hypothesis as the constraint satisfaction inference (CSInf) (Van Den Bosch and Canisius, 2006). The results in Table 3 (1-1+CSInf vs. 1-1+HMM) show that the HMM approach consistently improves performance over the baseline system (1-1 align), while the CSInf degrades performance on the Brulex data set. For the CSInf method, most errors are caused by trigram confusion in the prediction phase.

The results of our best system, which combines the HMM method with the many-to-many alignments (M-M+HMM), are better than the results reported in (Black et al., 1998) on both the CMUDict and German Celex data sets. This is true even though Black et al. (1998) use explicit lists of letter-phoneme mappings during the alignment process, while our approach is a fully automatic system that does not require any handcrafted list.

6 Conclusion and future work

We presented a novel technique of applying many-to-many alignments to the letter-to-phoneme conversion problem. The many-to-many alignments relax

Language	Data set	PRONALSYS	1-1 align	1-1+CsInf	1-1+HMM	M-M align	M-M+HMM
English	CMUDict	58.3 ± 0.49	60.3 ± 0.53	62.9 ± 0.45	62.1 ± 0.53	65.1 ± 0.60	65.6 ± 0.72
English	Celex	—	74.6 ± 0.80	77.8 ± 0.72	78.5 ± 0.76	82.2 ± 0.63	83.6 ± 0.63
Dutch	Celex	84.3 ± 0.34	86.6 ± 0.36	87.5 ± 0.32	87.6 ± 0.34	91.1 ± 0.27	91.4 ± 0.24
German	Celex	86.0 ± 0.40	86.6 ± 0.54	87.6 ± 0.47	87.6 ± 0.59	89.3 ± 0.53	89.8 ± 0.59
French	Brulex	86.3 ± 0.67	87.0 ± 0.38	86.5 ± 0.68	88.2 ± 0.39	90.6 ± 0.57	90.9 ± 0.45

Table 3: Word accuracies achieved on data sets based on the 10-fold cross validation. **PRONALSYS:** one-to-one alignments provided by the PRONALSYL challenge. **1-1 align:** our one-to-one alignment method described in Section 2.1. **CsInf:** Constraint satisfaction inference (Van Den Bosch and Canisius, 2006). **M-M align:** our many-to-many alignment method. **HMM:** our HMM embedded with a local prediction.

the constraint assumptions of the traditional one-to-one alignments. Letter chunking bigram prediction incorporates many-to-many alignments into the conventional phoneme prediction models. Finally, the HMM technique yields global phoneme predictions based on language models.

Impressive word accuracy improvements are achieved when the many-to-many alignments are applied over the baseline system. On several languages and data sets, using the many-to-many alignments, word accuracy improvements ranged from 2.7% to 7.6%, as compared to one-to-one alignments. The HMM cooperating with the local predictions shows slight improvements when it is applied to the many-to-many alignments. We illustrated that the HMM technique improves the word accuracy more consistently than the constraint-based approach. Moreover, the HMM can be easily incorporated into the many-to-many alignment approach.

We are investigating the possibility of integrating syllabification information into our system. It has been reported that syllabification can potentially improve pronunciation performance in English (Marchand and Damper, 2005). We plan to explore other sequence prediction approaches, such as discriminative training methods (Collins, 2004), and sequence tagging with Support Vector Machines (SVM-HMM) (Altun et al., 2003) to incorporate more features (context information) into the phoneme generation model. We are also interested in applying our approach to other related areas such as morphology and transliteration.

Acknowledgements

We would like to thank Susan Bartlett, Colin Cherry, and other members of the Natural Language Processing research group at University of Alberta for their helpful comments and suggestions. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

David W. Aha, Dennis Kibler, and Marc K. Albert. 1991. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. 2003. Hidden Markov Support Vector Machines. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*.

Harald Baayen, Richard Piepenbrock, and Leon Gulikers. 1996. The CELEX2 lexical database. LDC96L14.

Alan W. Black, Kevin Lenzo, and Vincent Pagel. 1998. Issues in building general letter to sound rules. In *The Third ESCA Workshop in Speech Synthesis*, pages 77–80.

Michael Collins. 2004. Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Langauge Processing (EMNLP)*.

Walter Daelemans and Antal Van Den Bosch. 1997. Language-independent data-oriented grapheme-to-phoneme conversion. In *Progress in Speech Synthesis*, pages 77–89. Springer, New York.

Walter Daelemans, Jakub Zavrel, Ko Van Der Sloot, and Antal Van Den Bosch. 2004. TiMBL: Tilburg Memory Based Learner, version 5.1, reference guide. In *ILK Technical Report Series 04-02*.

Robert I. Damper, Yannick Marchand, John DS. Marsters, and Alexander I. Bazin. 2005. Aligning text and phonemes for speech technology applications using an EM-like algorithm. *International Journal of Speech Technology*, 8(2):147–160, June.

Arthur Dempster, Nan Laird, and Donald Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. In *Journal of the Royal Statistical Society*, pages B:1–38.

Yannick Marchand and Robert I. Damper. 2000. A multistrategy approach to improving pronunciation by analogy. *Computational Linguistics*, 26(2):195–219, June.

Yannick Marchand and Robert I. Damper. 2005. Can syllabification improve pronunciation by analogy of English? In *Natural Language Engineering*, pages (1):1–25.

Eric Sven Ristad and Peter N. Yianilos. 1998. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532.

David Sankoff and Joseph Kruskal, 1999. *Time Warps, String Edits, and Macromolecules*, chapter 2, pages 55–91. CSLI Publications.

Terrence J. Sejnowski and Charles R. Rosenberg. 1987. Parallel networks that learn to pronounce English text. In *Complex Systems*, pages 1:145–168.

Paul Taylor. 2005. Hidden Markov Models for grapheme to phoneme conversion. In *Proceedings of the 9th European Conference on Speech Communication and Technology 2005*.

Kristina Toutanova and Robert C. Moore. 2001. Pronunciation modeling for improved spelling correction. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 144–151, Morristown, NJ, USA. Association for Computational Linguistics.

Antal Van Den Bosch and Sander Canisius. 2006. Improved morpho-phonological sequence processing with constraint satisfaction inference. *Proceedings of the Eighth Meeting of the ACL Special Interest Group in Computational Phonology, SIGPHON '06*, pages 41–49, June.