Rule-Based Learning of Word Pronunciations from Training Corpora

by

Lajos Molnár

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

Author ....
Department of Electrical Engineering and Computer Science
May 18, 1998

Certified by ....
Christopher M. Schmandt
Principal Research Scientist
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Rule-Based Learning of Word Pronunciations from Training Corpora

by

Lajos Molnár

Submitted to the Department of Electrical Engineering and Computer Science
May 18, 1998
In partial fulfillment of the requirements for the degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This paper describes a text-to-pronunciation system using transformation-based error-driven learning for speech-recognition purposes. Efforts have been made to make the system language independent, automatic, robust and able to generate multiple pronunciations. The learner proposes initial pronunciations for the words and finds transformations that bring the pronunciations closer to the correct pronunciations. The pronunciation generator works by applying the transformations to a similar initial pronunciation. A dynamic aligner is used for the necessary alignment of phonemes and graphemes. The pronunciations are scored using a weighed string edit distance. Optimizations were made to make the learner and the rule applier fast. The system achieves 73.9% exact word accuracy with multiple pronunciations, 82.3% word accuracy with one correct pronunciation, and 95.3% phoneme accuracy for English words. For proper names, it achieves 50.5% exact word accuracy, 69.2% word accuracy, and 92.0% phoneme accuracy, which outperforms the compared neural network approach.

Thesis Supervisor: Christopher M. Schmandt
Title: Principal Research Scientist

Acknowledgments

I would like to thank Texas Instruments for providing support for this research. In particular, I would like to thank the Media Technologies Laboratory, for providing a productive and welcoming atmosphere. My greatest gratitude is towards Charles Hemphill, who guided me through the entire project and suggested new goals and new point of views. Thanks, for giving me confidence, and not letting me be disappointed. I also want to recognize Rajeev Agarwal, Thomas Staples, John Godfrey, Yu-Hung Kao, Wally Anderson, Kazz Kondo, Lorin Netsch, Ramli Ramalingam, Yeshwant Muthusamy and Yifon Gong for their input and volunteering in my experiments. I am thankful for Vishu Viswanathan and Raja Rajasekaran for suggesting improvements to the system, and for pushing me to examine the different aspects of its performance. Christopher Schmandt for supervising this thesis and giving me useful and practical suggestions on making it better. Teresa Almaguer and Laura Key for their devotion to students, for organizing study breaks, and generally improving our determination about our projects. I am very grateful for my family, my mom, dad, brother, sister, grandma and aunt for their encouragement, emotional support, and forgiveness for my sudden lack of attention. I cherish you dearly. Grandpa, I wish you also could have seen my thesis come to completion! Lots of thanks to Gary Sisto, Michael Hooten, Craig Diehl, Michael Koelsch, Scott Murphy, John Maline and Julie Underwood for changing my world at TI completely. I will never forget you! Of course, I am obliged to my friends in Dallas, Michael Burt, Mickie Cheng, Phyllis Williams and John Trimble, who shared my joy of success and development. I love you all! I would also want to acknowledge Simon Lau, Christopher Tserng and Rushani Wirasinghe for helping me rationalize the project, seeing me complete it and not criticizing my pronunciation, or at least not constantly. I want to express my gratitude to LeeAnn Henn for her invaluable feedback and for her time spent proofreading my thesis. Thanks for the free Mountain Dews and Pepsis at Networks! Thanks for the food on Friday night! I also want to thank Rashmi Khare here, although she did not contribute to this thesis, only to my other one. My final thanks go to my friends in Boston, Yuying Chen, LeeAnn Henn and especially Michael Thelen for being there for me when I really needed. I could not have done it, at least not with so much enjoyment, without you all. And do not forget all the students at MIT, who — despite their frantic need for workstations — did not log me out when I left the computer for more than 30 minutes, even when I used an untimed screensaver.

Contents

List of Figures

List of Tables

Chapter 1

Overview

Speech recognizers are becoming an important element of communication systems. These recognizers often have to recognize arbitrary phrases, especially when the information to be recognized is from an on-line, dynamic source. To make this possible, the recognizer has to produce pronunciations for arbitrary words. Because of space requirements, speech systems need a compact yet robust method to make up word pronunciations.

This paper describes a text-to-pronunciation system built at the Texas Instruments Media Technologies Laboratory's Speech Recognition Group that is able to generate pronunciations for arbitrary words. The system extracts language-specific pronunciation information from a large training corpus (a set of word pronunciations), and is able to use that information to produce good pronunciations for words not in the training set.

In this chapter, we outline the purpose of the project, elaborate on a few methods currently used to generate word pronunciations, and give a brief reasoning for our approach, the transformation-based error-driven learner.

The error-driven learner was originally designed for part-of-speech (POS) tagging systems, and we adapted it to produce text-to-pronunciation (TTP) translations. Chapter 2 describes the core algorithm of the learner through the original POS tagging problem. It also elaborates on the similarities and differences between the problem of part-of-speech tagging and the problem of text-to-pronunciation translation.

Chapter 3 describes how transformation-based error-driven learning can be used for TTP systems, and what changes to the original POS tagging learner are necessary. This is where our design considerations are discussed.

Chapter 4 describes the system we built to capture pronunciation rules from training corpora. It also describes how the system generates pronunciations for arbitrary words. The final part of the chapter discusses some of our extensions to the original system, so that it can learn to produce multiple pronunciations and use phonetic features to generalize transformational rules.

In Chapter 5 we evaluate the system's performance and compare it to two other systems that are used to generate pronunciations. We also propose and discuss possible improvements and extensions to our system.

1.1 Notations

In this document we will frequently refer to phonemes and graphemes (letters). Graphemes are enclosed in single quotation marks (e.g. `'abc'`). In fact, any symbol(s) within single quotation marks refer to graphemes, or grapheme sequences.

Phonemes or phoneme sequences are enclosed in parentheses: *(' m uw)*. We will use the ASCII representation for the English phoneme set, as described in [7]. Stress levels are usually not marked in most examples, as they are not important to the discussion. In fact, we will assume that the stress information directly belongs to the vowels, so the above phoneme sequence will be denoted as *(m 'uw)* or simply *(m uw)*. Schwas are represented either as unstressed vowels *(.ah)* or using their special symbol *(ax)* or *(.ax)*.

Grapheme-phoneme correspondences (partial or whole pronunciations) are represented by connecting the graphemes to the phoneme sequence (e.g. `'word'` $\longrightarrow$ *(w er d)*) [1].

Grapheme or phoneme contexts are represented by listing the left and right contexts, and representing the symbol of interest with an underscore (e.g. *(b _ l)*). Word

[1]Grapheme-phoneme correspondences usually do not contain stress marks.

boundaries in contexts are denoted with a dollar sign (e.g. '\$x_').

1.2 Purpose of the project

1.2.1 Background

In this decade, speech as a medium is becoming a more prevalent component in consumer computing. Games, office productivity and entertainment products use speech as a natural extension to visual interfaces. Some programs use prerecorded digital audio files to produce speech, while other programs use speech synthesis systems. The advantage of the latter systems is that they can generate a broad range of sentences, and thus, they can be used for presenting dynamic information. Nevertheless, their speech quality is usually lower than that of prerecorded audio segments.

Speech recognition systems are also becoming more and more accessible to average consumers. A drawback of these systems is that speech recognition is a computationally expensive process and requires a large amount of memory; nonetheless, powerful computers are becoming available for everyday people.

Both speech synthesis and speech recognition rely on the availability of pronunciations for words or phrases. Earlier systems used pronunciation dictionaries to store word pronunciations. However, it is possible to generate word pronunciations from language-specific pronunciation rules. In fact, systems starting from the early stages have been using algorithms to generate pronunciations for words not in their pronunciation dictionary. Also, since pronunciation dictionaries tend to be large, it would be reasonable to store pronunciations only for words that are difficult to pronounce, namely, for words that the pronunciation generator cannot correctly pronounce.

1.2.2 Desired properties

The purpose of this project was to build a system that is able to extract pronunciation information from a set of word pronunciations, and is able to generate pronunciations for words not in the list. Other desired properties of the system are that it should be language independent and require minimal information about the language in question. Also, the pronunciation generation should be fast and use only a small amount of space, while the pronunciation information should also be captured compactly.

An important factor we need to consider is that when we generate pronunciations used for speech recognition, we want to generate all acceptable (and plausible) pronunciations, so that our system will be robust. (We will refer to systems that generate all plausible pronunciations for words as *text-to-pronunciation systems* in the rest of this document.) Since we need all plausible pronunciations, our system needs to be able to capture rules that can generate multiple pronunciations for a given word. Notice that this is not desirable when we generate pronunciations for speech synthesis, where we only want one pronunciation: the correct one.

1.2.3 Scope of the project

The purpose of text-to-pronunciation systems is to produce a phonetic representation of textual data. The scope of such systems can vary from producing pronunciations for single words to transcribing technical papers to a phonetic representation. In the most extreme cases, the input domain can contain made-up words, acronyms, numbers, symbols, abbreviations or even mathematical expressions.

In practice, high-end text-to-pronunciation systems transcribe text in two steps. First, the input text is normalized during which numbers, symbols, abbreviations, and expressions are written out into their full textual form. In addition, acronyms are expanded into separate letters or pseudo-words if those constitute plausible (pronounceable) words in the given language. During the second step, the phonetic representations of the written-out phrases are generated. In this project, we are only concerned with producing pronunciations for single words.

There are other aspects of text-to-speech conversion that are worth mentioning here. For many languages, such as Russian or English, morphological analysis is necessary to obtain the proper pronunciation of words. In Russian, for example, some vowels are reduced if they are unstressed, and the stress pattern of a word is determined by the word morphology. Some sort of lexical analysis (syntactical or morphological) is also required for the text normalization step [15]. For example, in English the phonetic form of '`$5`' is different in '`a $5 bill`' and in '`I got $5.`' In Russian, the case of nouns depends on their exact quantity; therefore, some sort of lexical analysis of quantifier structures is required to produce the full textual form of noun abbreviations. The necessity of lexical analysis in both steps has prompted some to combine the two phases of text-to-pronunciation conversion into one single process [15].

Languages are constantly evolving, and new concepts appear every day. Today it is not uncommon to use e-mail addresses as phrases in conversation. There is a tremendous morphological analysis going on when we pronounce technical names that contain abbreviations. It is also likely that text-to-pronunciation systems will face such pseudo-words. For example, when pronouncing '`bsanders@mit.edu`', '`bsanders`' will very likely be an input to the word pronunciation generator. While we could detect such pseudo-words statistically, as the grapheme sequence '`bsa`' is not common in word-initial position in English, we could also simply learn to insert the *('iy)* sound in the grapheme context '`$b_s`'.

1.3 Current technology

There are a myriad of approaches that have been proposed for text-to-pronunciation systems. In addition to using a simple pronunciation dictionary, most systems use rewrite rules which have proven to be quite well-adapted to the task at hand. Unfortunately, these rules are handcrafted; thus, the effort put into producing these rules needs to be repeated when a new language comes into focus. To solve this problem, more recent methods use machine-learning techniques, such as neural networks, decision trees, instance-based learning, Markov models, analogy-based techniques, or data-driven solutions [16] to automatically extract pronunciation information for a specific language.

In this section we review some of the approaches that we considered for the task. Unfortunately, it is difficult to objectively compare the performance of these methods, as each is trained and tested using different corpora and different scoring functions. Nevertheless, we will give an overall assessment of each approach and explain how we came to our decision on which method to use.

1.3.1 Pronunciation Dictionaries

The simplest way to generate word pronunciations is to store them in a pronunciation dictionary. The advantage of this solution is that the lookup is very fast. In fact, we can have a constant lookup time if we use a hash table. It is also capable of capturing multiple pronunciations for words with no additional complexity. The major drawback of dictionaries is that they cannot seamlessly handle words that are not in them. They also take up a lot of space ($O(N)$, where N is the number of words[2]).

1.3.2 Simple context-based system

A somewhat more flexible solution is to generate pronunciations for words based on their spelling. In a pronunciation system developed by ARPA[3], each letter (grapheme) is pronounced based on its grapheme context. An example for English would be to

$$\text{pronounce `e' in the context `\_r\$' as } \textit{(er)}. \tag{1.1}$$

The system consists of a set of rules, each containing a letter context and a phoneme sequence (pronunciation) corresponding to the letter of interest marked in bold. The representation of the above rule (1.1) would be:

$$\text{`}\underline{\text{e}}\text{r\$'} \longrightarrow \textit{(er)} \tag{1.2}$$

[2]$O(f(x))$ is the mathematical notation for the order of magnitude.
[3]Advanced Research Projects Agency

These pronunciation rules are generated by a human expert for the given language. The advantage of this system is that it can produce pronunciations for unknown words; in fact, every word is treated as unknown. Also, this method can encapsulate pronunciation dictionaries, as entire words can be used as contexts. Furthermore, this method can produce multiple pronunciations for words, since the phoneme sequences in the rules can be arbitrary. The disadvantage of the system is that it cannot take advantage of phonetic features; thus, it requires an extensive rule set. Also, a human expert is needed to produce the rules; therefore, it is difficult to switch to a different language. Moreover, it pronounces each letter as a unit, which seems counter-intuitive.

1.3.3 Rule-based transliterator

The rule-based transliterator (RBT) [8] uses transformation rules to produce pronunciations. It was written in the framework of the theory of phonology by Chomsky and Halle [4], and it uses phonetic features and phonemes. Rewrite rules are formulated as

$$\alpha \longrightarrow \beta/\gamma\_\delta, \tag{1.3}$$

which stands for

α is rewritten as β in the context of γ (left) and δ (right).

Here, α, γ, and δ can each be either graphemes or phonemes. Each phoneme is portrayed as a feature bundle; thus, rules can refer to the phonetic features of each phoneme. Rewrite rules are generated by human experts, and are applied in a specific order.

This method is similar to the simple context-based method described in 1.3.2. One improvement is that this system can make use of phonetic features to generalize pronunciation rules. Also, it can capture more complex pronunciation rules because applied rules change the pronunciations which become the context for future rules. The major disadvantage of this solution is that a human expert is still needed to produce the rules; thus, it is difficult to switch to a different language. Another disadvantage is that in contrast with the simple context-based model, this method cannot produce multiple pronunciations for words. Nevertheless, it can be extended to handle multiple pronunciations if we specify how contexts are matched when the phonetic representation is a directed graph and contains multiple pronunciations.

1.3.4 Transformation-based, error-driven learning

The transformation-based error-driven learner is an extension of the rule-based transliterator. This approach uses similar rewrite rules to produce pronunciations; however, it derives these rules by itself. Also, the context in the rewrite rules can contain corresponding graphemes and phonemes, as this extra information helps the error-driven learner to discover rules.

The learning process consists of the following steps. First, the spelling and the pronunciation of each word in the training set is aligned with each other. Then, an initial pronunciation is produced for each word. After that, the learner produces transformations that bring the pronunciation guesses closer to the true pronunciations. The most successful transformation is applied to the training set, and then the process is repeated until there are no more transformations that improve the word pronunciations.

This method, based on Eric Brill's part-of-speech tagging system [2], can achieve very high accuracy [8]. The main advantage of this approach is that it is completely automatic, and it needs only a little knowledge about the language (phoneme-grapheme mappings). The disadvantage is that it is hard to produce multiple pronunciations with it, and it is prone to overlearning, in which case it memorizes word pronunciations as opposed to extracting meaningful pronunciation rules.

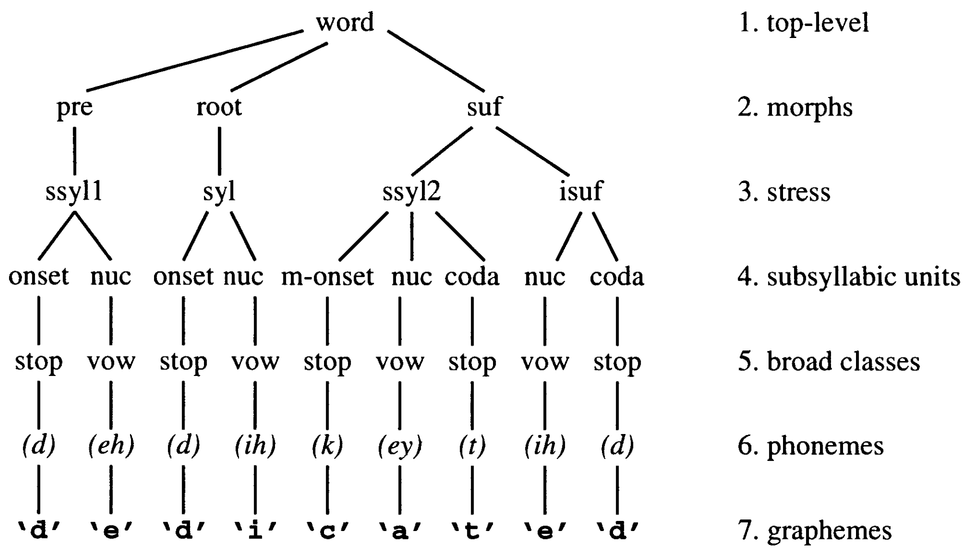
word
pre
root
suf
ssyl1
syl
ssyl2
isuf
onset nuc onset nuc m-onset nuc coda nuc coda
stop vow stop vow stop vow stop vow stop
(d) (eh) (d) (ih) (k) (ey) (t) (ih) (d)
'd' 'e' 'd' 'i' 'c' 'a' 't' 'e' 'd'
1. top-level
2. morphs
3. stress
4. subsyllabic units
5. broad classes
6. phonemes
7. graphemes

Figure 1-1: Linguistic layers used by a morphology-based TTP system.

1.3.5 Parsing word morphology

So far, we have not used any information — besides phonetic context — to produce word pronunciations. As one would expect, word morphology can have a large effect on word pronunciations, especially when it comes to stress. Predicting unstressed syllables is important for speech recognition, as these vowels tend to have very different characteristics than their stressed counterparts. The Spoken Language Systems Group at MIT proposed a system that uses word morphology information to generate word pronunciations with stress information [10].

In their method, they generate a layered parse tree (shown in Figure 1-1) to examine several linguistic layers of the words in the training set. Then, they examine various conditional probabilities along the tree, such as the probability that a symbol follows a column, etc. During pronunciation generation, they try to generate a parse tree for the word while maximizing its overall probability. The advantage of this method is that it generates very accurate pronunciations while also producing morphological structure. However, it needs the full morphological structure of words in the training set, which can be very expensive to provide when training for a new language. Also, this method cannot produce multiple pronunciations in its current form.

1.3.6 Overlapping Chunks

The overlapping chunks method tries to relate to human intelligence, as it mimics how people pronounce unseen words. The method uses multiple unbounded overlapping chunks to generate word pronunciations. Chunks are corresponding grapheme and phoneme sequences that are cut out from word pronunciations. For example, `'anua'` $\longrightarrow$ *('ae n y .ah w .ah)* is a chunk derived from `'manual'` $\longrightarrow$ *(m 'ae n y .ah w .ah l)*. In this method, first, all possible chunks are generated for the words in the knowledge base. When a new pronunciation is requested, these chunks are recombined in all possible ways to produce the requested word. During this process, chunks can overlap if the overlapping phonemes and graphemes are equivalent. After finding all possible recombinations, the best pronunciation candidate is selected. In general, candidates with fewer and longer chunks are favored, especially if those chunks are largely overlapping.

The advantage of this system is that it is language independent, and it can truly produce multiple pronunciations. Also, very little language specific information is needed aside from the word pronunciations, although the words have to be aligned with their pronunciations. The main disadvantage of the system is that it requires a lot of run-time memory during pronunciation generation to speed up the recombination process.[4]

1.3.7 Neural networks

Neural networks are used in many areas of automated learning, including to generate word pronunciations. The most popular network type is the *multilayer perceptron network* (MLP)[11] where the processing units are arranged in several layers, and only adjacent layers are connected. During processing, activations are propagated from the input units to the output units. There is one input unit at each grapheme space for

[4]There are other theoretical deficiencies of this algorithm, which are described in [16].

each possible grapheme, and similarly, there is one output unit at each phoneme slot for every phoneme of the language. During pronunciation generation, the input units at the appropriate graphemes of the words are activated. The pronunciation is composed from the phonemes by the output units holding the largest activation value at each phoneme slot. Neural networks are trained using an iterative *backpropagation algorithm.*

The advantages of this approach are that it is language independent, very little language-specific side information is needed, and it can produce multiple pronunciations. One can further improve this method by also assigning input units to phonetic features, so that it can make use of phonetic features. The disadvantage of this method is that the resulting neural network is large. Also, it is hard to capture the phonetic knowledge efficiently from the activation properties. For example, it is hard to separate the important information from the non-important information to balance the neural network size and performance.

1.4 Our approach

An overview of the complexities and performances of the different approaches is shown in Table A.2 on page 79. Our goal was to find an automatic method that is fast, uses a small amount of space, and is effective at producing correct pronunciations for words. Only the last four systems are truly automatic, and our space requirement eliminated the overlapping chunks approach. We chose the transformation-based error-driven learner because we did not have word morphology information, and because it has a smaller memory requirement during pronunciation generation than the method using neural networks.

We could have chosen other rule-based systems that share similar qualities as the transformation-based learner. Substitution and decision-tree based approaches are also used successfully to resolve lexical ambiguity [5, 6]. The benefit of Brill's system is that it is completely deterministic: rules are applied in a specific order. In the other systems rules are competing against each other, which greatly increases translation time. Also, in Brill's system the most relevant rules have the highest rank; therefore, much of the accuracy is achieved with the first few hundred rules [1, 2]. This allows us to compromise between translation time and pronunciation accuracy.

Chapter 2

Transformation-Based Error-Driven Learning

Before we illustrate how transformation-based error-driven learning is used for text-to-pronunciation (TTP) systems, it is important to see how it was originally applied to part-of-speech (POS) tagging. In this chapter, we describe the core of the learning algorithm in a case study for part-of-speech tagging. We also discuss the similarities and differences between POS tagging and TTP systems.

2.1 A similar problem: part-of-speech tagging

Both part-of-speech tagging and text-to-pronunciation systems try to resolve lexical ambiguity. In a POS tagging system, we want to find the part of speech for each word in a sentence. Each word has a list of possible parts of speech, but usually only one of them is realized in a sentence. For example, take the sentence:

$$\text{He leads the group.} \tag{2.1}$$

The possible part-of-speech tags are listed for each word in Table 2.1; the sentence (2.1) above is tagged the following way:

Word	Part of Speech
group	N, V
he	N
leads	N, V
the	det

Table 2.1: Possible part-of-speech tags for words in the sentence 'He leads the group.'

$$\begin{tikzcd} \text{He} & \text{leads} & \text{the} & \text{group} \\ \text{N} \arrow[u] & \text{V} \arrow[u] & \text{det} \arrow[u] & \text{N} \arrow[u] \end{tikzcd}$$

In a text-to-pronunciation problem we have a word composed of a sequence of letters (graphemes), and we want to map that sequence into the pronunciation — a sequence of phonemes. An intuitive solution would be to map each letter into its pronunciation, since we pronounce words based on their spelling. In this case, we can assume that each letter has several possible ways of being pronounced, and we have to find the one that is actually realized in the pronunciation. For example, to produce the pronunciation for

$$\texttt{`computer'}, \tag{2.2}$$

we need to select, for each letter, the proper phoneme sequence from the possible sequences shown in Table 2.2.

The realized pronunciation is:

$$\begin{tikzcd} \texttt{`c'} & \texttt{`o'} & \texttt{`m'} & \texttt{`p'} & \texttt{`u'} & \texttt{`t'} & \texttt{`e'} & \texttt{`r'} \\ (k) \arrow[u] & (ah) \arrow[u] & (m) \arrow[u] & (p) \arrow[u] & (y\ uw) \arrow[u] & (t) \arrow[u] & () \arrow[u] & (er) \arrow[u] \end{tikzcd}$$

Eric Brill's transformation-based error-driven learner [2] was found efficient to resolve lexical ambiguities of these sorts. The remainder of this chapter describes the transformation-based error-driven learner in view of part-of-speech tagging. In

Grapheme	Phoneme Sequences
`c`	*(k)*, *(s)*
`o`	*(ah)*, *(ow)*, *(aa)*, *(aw)*, *(ao)*, *(uw)*
`m`	*(m)*
`p`	*(p)*
`u`	*(ah)*, *(y ah)*, *(y uw)*, *(uw)*
`t`	*(t)*
`e`	*(ih)*, *(iy)*, *(eh)*, *(ah)*, *()*
`r`	*(r)*, *(er)*.

Table 2.2: Possible pronunciations of individual letters in the word `'computer'`.

addition to text-to-pronunciation, this method can also be applied to other problems with lexical ambiguity, such as syntactic parsing [1, 13], prepositional attachment, spelling correction [9], or grapheme-to-phoneme conversion [8].

2.2 Other methods for part-of-speech tagging

Methods for POS tagging evolved similarly to those for TTP translation. In fact, some of the approaches we examined in section 1.3 are also used for part-of-speech tagging. The first part-of-speech taggers used hand-written rules, and annotated text with 77% accuracy [14]. Once tagged corpora became available, statistical approaches could extract probabilistic information to achieve better performance. Neural networks achieve 86-95% tagging accuracy, depending on the size of the training set [14]. The accuracy of Markov and Hidden Markov models is around 96% [14]. These statistical methods capture tagging information with a large number of weights (activation values or probabilities).

In this decade, attention turned to rule-base systems because it is easier to understand the meaning of rules than the implication of probabilistic information. Brill's transformation-based error-driven learner, also a rule-based system, achieves 97% tagging accuracy [2]. Other rule-based systems, such as decision trees and substitution-based learners [5] are also effective in resolving lexical ambiguity. Nevertheless, transformation-based learners are superior to both alternatives: they can encapsulate the behavior of decision trees, and they overperform substitution-based systems because they are deterministic.

2.3 Transformation-based error-driven learner

This section describes the transformation-based error-driven learner for the case of part-of-speech tagging. The problem we are trying to solve is to tag a sentence with part-of-speech information. We assume that we have a large set of sentences with proper tagging (we will refer to it as *the truth*), and we want to find tagging rules so that we can reproduce the correct part-of-speech information from scratch.

The main mechanism in the transformation-based error-driven learner is rule induction. Rule induction takes a preliminarily tagged corpus as an input (referred to as *proposed solution*), and it finds a transformation rule that brings the tagging closer to the truth. The main idea is that if we iteratively apply this step, we get a sequence of rules that brings the tagging closer and closer to the correct tagging.

These are the steps of the transformation-based error-driven learner:

1. First, we tag the words in the training set based on some scheme. This will be initially the proposed tagging.

2. We score the proposed solution by comparing it to the truth, and find a transformation rule that brings the proposed tagging closer to the correct tagging.

3. If we find a rule, we save it and apply it to the entire training set. The resulting tagging will be the proposed solution for the next step. We then repeat from step 2.

4. We stop when we cannot find a rule that improves the score, or we reach the truth. At that point the learner cannot improve the score any more.

There are various pieces of information that need to be further specified for the system. They are

- the initial tagging scheme (*initial state*),

- the *scoring function* we use to compare the proposed solution to the truth,
- the space of allowable transformations, and
- the rule selection strategy (which rule we pick from the ones that improve the score).

2.3.1 Initial State

The initial state is not specified by the learner. We can start by assigning the most popular tag to each word or even by tagging every word as noun. The only important thing to look out for is that the learner will learn transformation rules to correct the initial tagging; therefore, when we apply the learned rules to untagged sentences, these sentences will have to be in the corresponding initial state for the rules to make sense. Therefore, it is preferred that the initial state is deterministic.

The fact that the initial state is not specified makes it possible to use this learner as a post-processor to correct the tagging solutions proposed by other systems. For example, we might initially tag sentences with an existing part-of-speech tagger, and use this system to find transformation rules that improve the performance of the original tagger.

2.3.2 Scoring Function

The main property of the scoring function is that it encapsulates the extent of the difference between the proposed solution and the truth. The specifics of the scoring function are only important if we have a strategy for choosing among the rules that improve the score. Brill's learner always picks the rule that improves the score the most. This strategy is useful, as this way the first rules will have the biggest impact on the correctness and the transformation rules will be ordered based on their importance on the performance.

It is important to choose the scoring function in a way that the penalties for deviations between the truth and the proposed solutions are in line with the importance of these mistakes. As an example, a noun tagged as an adjective may not be as bad a mistake as a noun tagged as a verb.

2.3.3 Possible transformations

The space of possible transformations is potentially infinite. However, if we want the learner to have any practical value, we need to limit the space of allowable transformation. Brill uses rule templates to achieve this. A rule template is a rule where the specifics are not filled out, such as:

Change tagging T_1 to T_2 if the following word is w_1.

Change tagging T_1 to T_2 if the following word is tagged T_3.

When the learner looks for rules, it tries to fill out these templates with specific values and see how the resulting rules improve the score. Unfortunately, there are a vast number of ways to fill out the rule templates, yet only a small percentage of them makes sense. To solve this problem, the search for rules is data-driven. When the learner compares the proposed solution to the truth, it sees where they deviate from each other. Therefore, the learner at each mistake can propose rules that correct that particular mistake. Note that there can be several rules that correct a given mistake.

We still need to evaluate the effectiveness of each proposed transformation rule. Notice that a rule which was proposed to correct a mistake does not always improve the overall score because it applies at all locations that have the required context, and not just where the rule was proposed. Therefore, it is possible that a rule will lower the score at some of the places where it applies.

To evaluate the effect of a rule, we need to find out how it influences the score at each location where it applies. Fortunately, we can do that very efficiently if we keep some statistical information around.

To see how some statistical information can speed up rule evaluation, suppose our scoring function is simply the number of words with correct tagging. The rule we are trying to evaluate is:

Change tag T_1 to T_2 in context C.

For this rule the change in the score is:

$$\Delta score = N(T_1, T_2, C) - N(T_1, T_1, C) \tag{2.3}$$

where $N(T_{proposed}, T_{correct}, C)$ is the number of words in context C that are tagged as $T_{proposed}$ and should be tagged as $T_{correct}$.

Therefore, if we have the contextual statistics available, the evaluation of a rule's effectiveness takes only a very short, constant time.

2.4 Differences between part-of-speech tagging and text-to-pronunciation translation

In the previous section we depicted how transformation-based error-driven learning can be used for part-of-speech tagging. In this section we elaborate on the similarities and differences between the POS tagging and the TTP translation problem.

In a text-to-pronunciation system, we try to translate the word spelling (letters) to the word pronunciation (phonemes). While some languages have one-to-one correspondence between letters and phonemes, in most languages phonemes are only loosely associated with letters. An example in English is the word `'shootable'`, which is pronounced as *(sh 'uw t .ah b .ah l)*. If we were to have one-to-one correspondence between letters and phonemes (or even phoneme sequences), we would end up with many possible alternatives (Figure 2-1).

The main problem is that in many languages certain phonemes are represented by a group of graphemes. Also, phoneme insertions and deletions are natural phonetic processes that have to be accounted for. Therefore, mappings are better characterized as group-to-group, meaning mapping a group of phonemes to a group of graphemes (Figure 2-2).

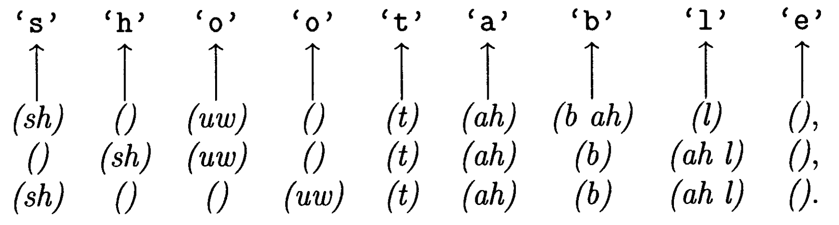
's' 'h' 'o' 'o' 't' 'a' 'b' 'l' 'e'
(sh) () (uw) () (t) (ah) (b ah) (l) (),
() (sh) (uw) () (t) (ah) (b) (ah l) (),
(sh) () () (uw) (t) (ah) (b) (ah l) ().

Figure 2-1: Possible phoneme-to-grapheme alignments for the word `'shootable'`.

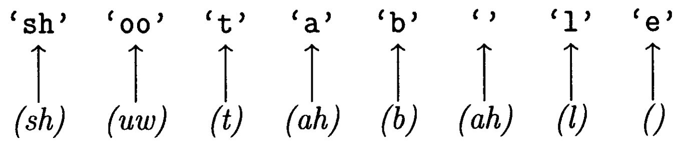
'sh' 'oo' 't' 'a' 'b' '' 'l' 'e'
(sh) (uw) (t) (ah) (b) (ah) (l) ()

Figure 2-2: Phoneme-group to grapheme-group alignment for the word `'shootable'`.

Unfortunately, these groups are not predetermined in a given language. Graphemes in a word can be assigned to plausible groups in many ways depending on word morphology and grapheme context. For example, the grapheme chunks `'sc'` and `'ph'` are grouped differently in `'s|ch|oo|l'` and `'sc|e|n|e'`, or `'l|oo|p|h|o|l|e'` and `'ph|y|s|i|c|s'` (Figure 2-3).

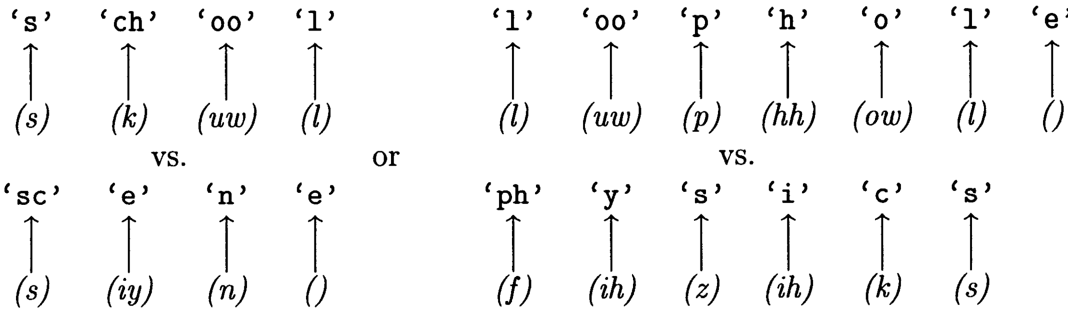

Figure 2-3: Different groupings of chunks `'sc'` and `'ph'` in English.

This means that groups need to possibly be reassigned during the learning process; therefore, we need rule templates that do reassignment. Another potential problem with the group-to-group assignments is that if group assignments are automatic, they may grow without bounds, rendering the alignments useless (Figure 2-4).

One solution is to assign phonemes to only one of the letters of the matching

$$\begin{tikzcd} \texttt{`scene'} \\ (s\ iy\ n) \arrow[u] \end{tikzcd}$$

Figure 2-4: Overgrown phoneme-grapheme alignments do not have any useful information.

$$\begin{tikzcd} \texttt{`s'} & \texttt{`c'} & \texttt{`h'} & \texttt{`o'} & \texttt{`o'} & \texttt{`l'} & & \texttt{`s'} & \texttt{`c'} & \texttt{`e'} & \texttt{`n'} & \texttt{`e'} \\ (s) \arrow[u] & (k) \arrow[u] & & (uw) \arrow[u] & & (l) \arrow[u] & & (s) \arrow[u] & & (iy) \arrow[u] & (n) \arrow[u] & \end{tikzcd}$$

Figure 2-5: The phoneme-grapheme alignment used in our system.

group. This would basically revert the alignment to the one in Figure 2-1. To avoid the problem mentioned there, we need to decide which phoneme is assigned to which letter in the group. Then, the only required transformations are the insertion, deletion, or modification of phonemes, as in this solution phonemes are always assigned to a single grapheme (Figure 2-5). We chose this solution because it simplifies the rule templates, and it is deterministic; thus, it proposes more coherent rules during rule induction.

Chapter 3

Design considerations

In the previous chapter we examined how the transformation-based error-driven learner is used to resolve lexical ambiguity in the field of part-of-speech tagging. We also showed that text-to-pronunciation translation is similar to the part-of-speech tagging problem, with the exception that in a TTP system we assign phonemes to the individual letters. We also discussed that the main difference between TTP and POS tagging systems is that the number of phonemes can change during the learning and applying process; therefore, we have to keep track of which phoneme belongs to which letter. In this chapter we discuss how transformation-based error-driven learning can be used for text-to-pronunciation translation, and we examine the various design decisions we have to make.

Section 2.3 on page 25 described the transformation-based error-driven learner in detail. To apply the learner to the problem of text-to-pronunciation translation, we need to specify:

- the initial state,
- the allowable transformation templates, and
- the scoring function.

3.1 Initial state

To minimize the amount of work that needs to be done during learning, we chose to assign the most probable phoneme sequence to each grapheme as the initial state. This way, the initial guesses will be closer to the truth should we use a simple string edit distance. We could also use a somewhat more intelligent approach that handles letter groups, such as `'ph'` or `'ch'`. Unfortunately, these groups can potentially conflict with one another as discussed in Section 2.4 on page 28, so we avoided this issue, especially since the context-based rules can encapsulate the effect of these letter groups.

In either case, we need to find the conditional probabilities of graphemes and phoneme sequences. There are two ways of getting this information. Either a human expert can input the grapheme-phoneme sequence correspondences for a given language, or we can try to automatically extract this information from the training corpora. The latter method is preferred as it will result in a more language independent system.

We can obtain the conditional probabilities of grapheme-phoneme correspondences if we align the words with their pronunciations. For that, we can use a probabilistic aligner given that we have some approximate values for the conditional probabilities themselves. While this seems like a vicious circle, the real value of this cycle is that if we have some approximate probabilities, we can use the probabilistic aligner to obtain better probability estimates. Furthermore, we can repeat this process to get better and better estimates, until the conditional probabilities are stable. Nevertheless, the most probable alignment is not necessarily the correct alignment; therefore, this method can never perfectly obtain the grapheme-phoneme conditional probabilities.

We found two ways to find a reasonable initial estimate for these probabilities. One of them — *the slice algorithm* — requires no additional information besides the word pronunciations, while the other requires the knowledge of some basic language-specific information for phonemes and letters.

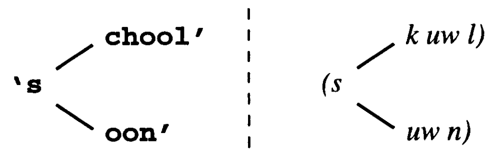

Figure 3-1: Phoneme-grapheme chunk from contrasting `'school'` with `'soon'`.

3.1.1 Slice algorithm

The slice algorithm can make reasonable guesses on the probabilities of grapheme-phoneme correspondences. The algorithm relies on the observation that the pronunciations of words that start similarly also start similarly, and one can find phoneme and grapheme units of the language by finding contrasting examples: words that start identically, but differ very soon after both in their spellings and their pronunciations.

An example of contrasting words in English is `'school'` and `'soon'`, illustrated in Figure 3-1. By contrasting `'school'` with `'soon'` we can find the following corresponding phoneme-grapheme chunk: (`'s'` $\longrightarrow$ *(s)*).

Using the notion of contrasting word pairs, the slice algorithm works the following way. It looks through the words in the training set, and finds words that have the same few initial letters and phonemes in their pronunciations. It tries to find contrasting pairs that differ both in pronunciation and spelling shortly after the beginning. If it finds sufficiently many contrasting pairs that differ at the same locations, it assigns the identical initial letters to the identical initial phoneme sequence. These segments are sliced off the words and their pronunciations, and the resulting truncated words and pronunciations are placed back to the training pool. Figure 3-2 illustrates the process of successive slicing.

Sooner or later there will be no contrasting word pairs. When the program cannot propose a new corresponding phoneme-grapheme pair, it tries to use the discovered grapheme-phoneme alignments to further slice off fragments from the words. In this process (referred to as *shredding*) the algorithm tries to place up to three grapheme-phoneme alignments along the beginning of the words while allowing graphemes to map to no phoneme. If there are grapheme and phoneme slices between the found

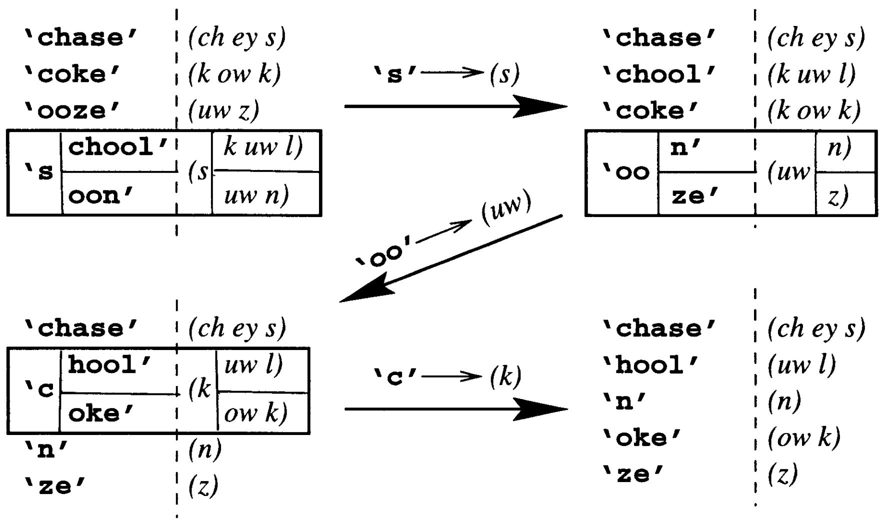

Figure 3-2: Successive slicing in the slice algorithm.

chunks, they are proposed as corresponding chunks as well (Figure 3-3).

The algorithm also automatically assigns one letter words to their pronunciations, and drops words in which the letters are completely sliced off, even if they have remaining phonemes. Once all the words have been shredded, the program returns to the slicing process. The iteration of alternating shredding and slicing ends when all the words completely disappear, or when shredding has no effect.

The reason for the slice algorithm is to have a language independent way of finding phoneme-grapheme correspondences in large pronunciation corpora without any additional language-specific knowledge. Nevertheless, the algorithm is not completely language independent. One problem area is that the algorithm is not symmetric; it slices the words at their beginning. Therefore, it is not useful for languages that have irregular pronunciations near the beginning of words — such as silent letters or seemingly random letter-pronunciations — although we do not know of any such language.

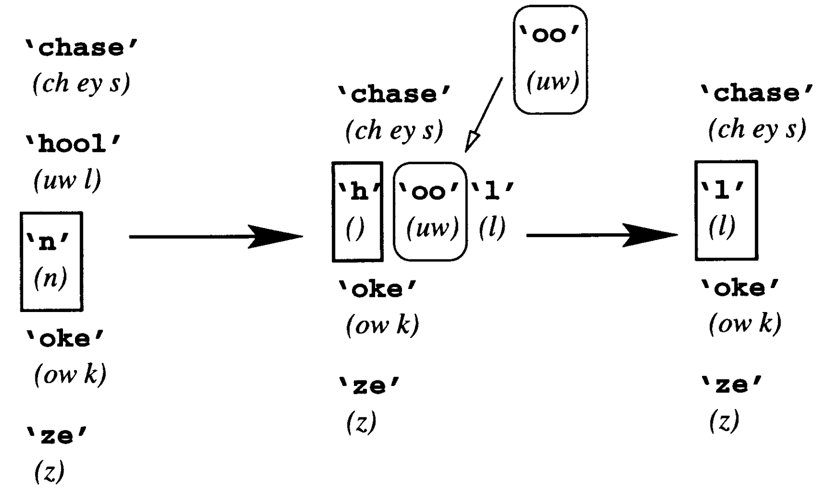

Figure 3-3: Using discovered grapheme-phoneme pairs to find new pairs in the slice algorithm (shredding). The previously discovered 'oo' $\longrightarrow$ *(uw)* alignment is used to discover the 'h' $\longrightarrow$ *()* chunk. One letter chunks, 'h' $\longrightarrow$ *()* and 'l' $\longrightarrow$ *(l)* are automatically assigned.

3.1.2 Phonetic information through other methods

Another way of finding grapheme-phoneme correspondences is outlined in [8]. This approach assumes that we know the vowel/consonant qualities for both phonemes and letters of the language. With that information, we align the words with their pronunciations while minimizing the vowel-consonant differences between the corresponding letters and phonemes. This approach gives better initial guesses while requiring slightly more phonetic knowledge than the slice method. Thus, a question naturally arises: should we add more phonetic information to achieve better accuracy?

Our design goals described in Section 1.2.2 state that we want to require as little language-specific side information as possible. While doing a preliminary alignment based on vowel/consonant quality gives better initial guesses for the conditional probabilities of grapheme-phoneme pairs, the probabilities after the iterative alignment process are basically identical. Therefore, we chose to use the slice method in our system.

3.2 Grapheme-phoneme alignment

We need to align graphemes to phonemes to find the conditional probabilities of grapheme-phoneme correspondences. We need these probabilities to come up with initial pronunciations for the words by assigning the most probable phoneme sequences to each letter (grapheme). However, there is also another, more compelling reason for aligning the phonemes with the graphemes.

In part-of-speech tagging systems, tags are automatically "aligned" with the words they belong to. We make use of this alignment, since the rule templates used for the part-of-speech tagging system refer to both the preceding/following words and their tags. In a text-to-pronunciation system the alignment between graphemes and phonemes is not obvious, but we nevertheless need this correspondence to refer to both adjacent graphemes and phonemes. Therefore, we must keep the graphemes and phonemes aligned with each other throughout the learning and pronunciation generation process.

3.2.1 Two alignment processes

We distinguish two different alignment processes. When we are finding the conditional probabilities of grapheme-phoneme sequence pairs, we align grapheme sequences to phoneme sequences. However, during the learning process we assign phonemes individually to graphemes as discussed in Section 2.4. The difference between the two alignments is illustrated in Figure 3-4.

The need to assign single phonemes to single graphemes results in some ambiguity if the original phoneme or a phoneme sequence was aligned to a group of graphemes. To avoid performance degradation due to random selection between the possible assignments, we assign each individual phoneme to the grapheme with which it has the highest conditional probability. If the probability is the same for several graphemes, we assign the phoneme to the leftmost of those graphemes (Figure 3-5).

For example, `'school'` is always aligned as shown on Figure 3-5, and not any other way, because the `'c'` $\longrightarrow$ *(k)* alignment has higher probability than the `'h'` $\longrightarrow$ *(k)*,

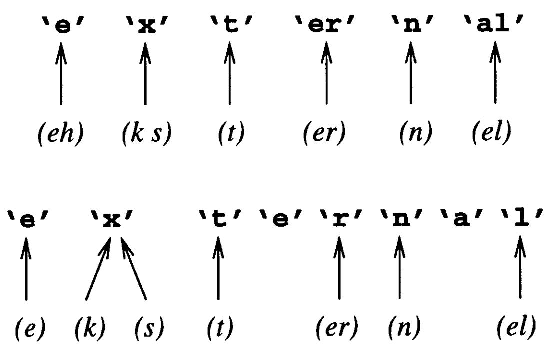
'e' 'x' 't' 'er' 'n' 'al'
(eh) (k s) (t) (er) (n) (el)
'e' 'x' 't' 'e' 'r' 'n' 'a' 'l'
(e) (k) (s) (t) (er) (n) (el)

Figure 3-4: The two different grapheme-phoneme alignments used during preprocessing. The top one is used during the iterative alignment process to update the conditional grapheme-phoneme probabilities. The bottom one is used by the learner.

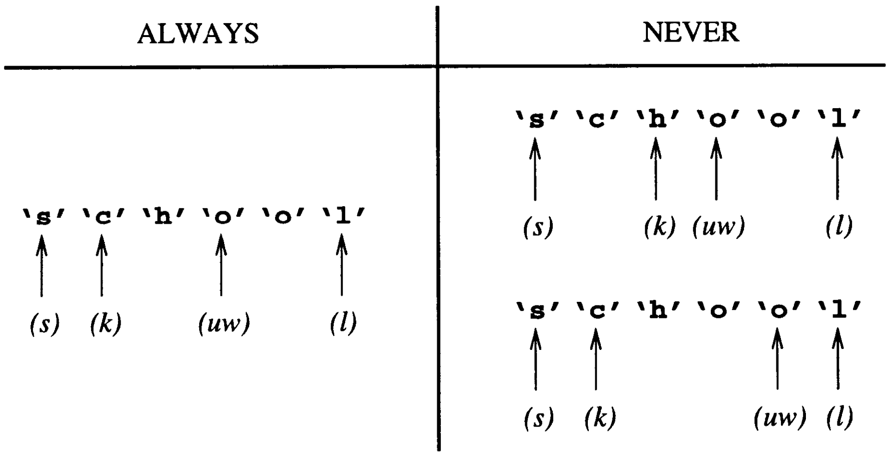

Figure 3-5: Allowed and disallowed alignments for 'school'. Phonemes are assigned to the first most probable grapheme (letter).

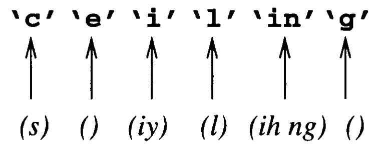

Figure 3-6: Misalignments that the iterative aligner cannot correct.

and *(uw)* is always aligned with the first `'o'`.

The aligner in our system has two corresponding phases. In the first phase phonemes or phoneme sequences are aligned with graphemes or grapheme groups. This is done by a dynamic aligner that maximizes the overall probability of the alignment based on the individual probabilities.

If we need to align the phonemes individually with the letters, phonemes or phoneme sequences that are aligned with a grapheme group are distributed among the graphemes using another dynamic aligner, which maximizes the joint probabilities for the individual phonemes in the sequence. If this dynamic aligner encounters a tie in the probabilities, it prefers the alignment where more phonemes are assigned to graphemes closer to the beginning of the word.

3.2.2 Is the iterative aligner the best?

It is questionable whether our aligner with iterative probability reassessment achieves correct grapheme-phoneme correspondence. If the initial alignments were conceptually wrong, the iterative aligner would not be able to correct these mistakes. Figure 3-6 illustrates a possible scenario, where the slice algorithm aligned `'in'` with *(ih ng)*. Although it makes more sense to align `'i'` with *(ih)*, and `'ng'` with *(ng)*, the aligner has no means of making such observation, since maximizing the joint probabilities will reinforce the problem. This issue could be solved by experimenting with the location of phoneme-group and grapheme-group boundaries in common neighbors to see if a different boundary would make the alignments more general.

Another related problem is that aligning `'ei'` with *(iy)* would be better than aligning `'e'` and `'i'` separately. This problem could be resolved similarly, by investigating the effects of grouping common neighbors, where one grapheme maps to an empty pronunciation.

It is also questionable whether the alignment with maximal joint probability is the proper alignment. In an experiment, we hand aligned 1000 words with their pronunciations, and calculated the conditional grapheme-phoneme sequence probabilities. We then aligned the pronunciations using these probabilities and found that the phoneme-alignment error was 1.5%, with a word error rate of 3.6%.

We also aligned these words using the probabilities produced by the slice algorithm, and we got a comparable 3.0% phoneme-alignment error and 14% word error. When we examined the alignment errors, we found that in 7.9% of the words the alignments were not necessarily bad. These word errors resulted from either a different, but correct alignment (5.3%), or an alignment that was simply different, but not wrong (2.6%). The other 6.1% of the words had correctable misalignments (2.5%) or major discrepancies (3.6%).

In summary, while iterative alignment may not achieve perfect alignments due to its nature, it is effective in correcting approximate grapheme-phoneme sequence probabilities. Also note that by the time the alignments are used in the learner, all correctable mistakes are corrected. Therefore, the visible word error rate of the slice algorithm with iterative alignment is only 3.6%.

3.3 Scoring proposed pronunciations

We used a simple string edit distance [12] to calculate the correctness of the proposed solution. The edit distance is the minimal number of phoneme insertions, changes, and deletions needed to bring the proposed solution to the truth. We chose to penalize all three transformations with the same weight, except that stress mistakes were taken only half as seriously.

Since the transformation-based error-driven learner is data-driven — meaning that only rules that correct mistakes are suggested — the scoring function needs to be able to produce feedback on how the proposed pronunciation has to be corrected.

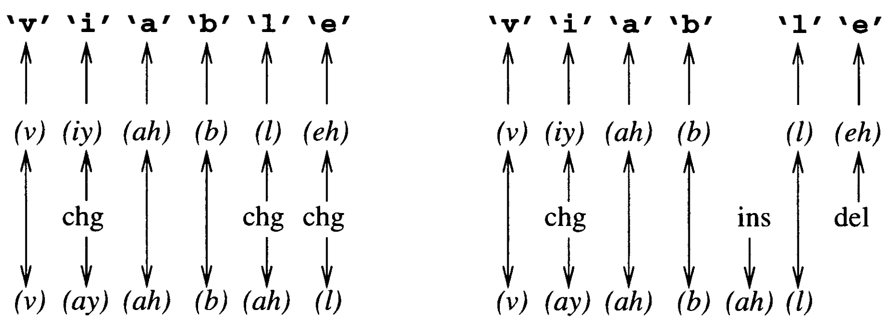
'v' 'i' 'a' 'b' 'l' 'e'
(v) (iy) (ah) (b) (l) (eh)
chg
chg chg
(v) (ay) (ah) (b) (ah) (l)
'v' 'i' 'a' 'b' 'l' 'e'
(v) (iy) (ah) (b) (l) (eh)
chg
ins
del
(v) (ay) (ah) (b) (ah) (l)

Figure 3-7: Non-uniform alignments of the same disparity.

3.3.1 Phoneme-phoneme alignments

We used a dynamic aligner to minimize the string edit distance between our guesses and the truth, which unfortunately does not produce uniform alignments. This means that if we have the same non-matching phoneme sequences in two word pairs, the resulting alignments of the non-matching parts are not necessarily the same (Figure 3-7). Consequently, if we propose rules from these alignments, we are not getting a uniform picture on each mistake.

We solved this problem by adding phonetic knowledge to the system. We changed the distance function used in the alignments so that at phoneme substitutions, phonemes with similar phonetic qualities (both are vowels, both are stops, etc.) are preferred to be aligned with each other. Here we relied on the assumption that even if there is a disparity between the proposed and the correct pronunciation, the differing phonemes have similar phonetic features.

3.3.2 Problems with using a simple string edit distance for scoring

There is a potential problem with using simple string edit distance as an error measure; namely, that it is probably not correct to weigh every phoneme error equally. For example, our distance measure should weigh phoneme differences that are not very contrasting less heavily if we want to use the generated pronunciations in a speech recognizer.

We ran an experiment to investigate this problem. In the experiment we compared two sets of pronunciations. We asked human subjects (speech recognition experts) to rate each pronunciation on a three-point scale (1.00-3.00). Surprisingly, the set with the better edit score (84%) was rated uniformly lower by the subjects (2.00). The other set with 79% edit score was rated at 2.25. Nevertheless, we did not change the error measure in our experiment because requiring relative seriousness of phoneme differences would violate our desire to use minimal linguistic knowledge and be automatic and language independent.

Chapter 4

The system

This chapter describes the text-to-pronunciation system we have developed. The system has two major parts, the learner and the rule applier (pronunciation generator), as illustrated by Figure B-1 on page 82. The pronunciation learner extracts pronunciation rules from a training corpus and supplemental phonetic information, and the rule applier applies the resulting rules to produce pronunciations for new or unseen words.

4.1 Learning text-to-pronunciation rules

Learning the pronunciation rules consists of five steps. The first four steps are mere preprocessing, the actual learning takes place during the fifth step. The steps are:

1. Finding initial approximations for the grapheme-phoneme conditional probabilities using the slice algorithm.

2. Finding more precise grapheme-phoneme conditional probabilities using iterative probabilistic (re)alignment of grapheme and phoneme sequences.

3. Preparing for learning. Aligning individual phonemes with graphemes in the training set.

4. Initializing for learning. Proposing an initial pronunciation guess for each word.

5. Finding pronunciation rules

For the learning process, we need the following language-specific information:

- a large set of word pronunciations, and
- phonetic features, their importance and feature descriptions for all phonemes.

We do not need to know all phonetic features for a specific language, just the important ones, such as consonant/vowel quality, place of articulation, stop-fricative quality, etc. The learner works with a very small set of phonetic features as well, although adding features tends to improve the performance of the system.

4.1.1 Preprocessing

Initial probabilities

Finding initial approximations for the grapheme-phoneme conditional probabilities is done using the 'slice algorithm'. This method requires no linguistic information besides the word pronunciations. The output of the slice algorithm tends to be very noisy because the algorithm proposes rules as it sees them, and one bad proposition usually results in a whole set of bad propositions. This problem is illustrated by Figure 4-1, where by the end of the slicing, only 40% of the remaining word chunks are consistent with their pronunciations. Suggesting corresponding phoneme-grapheme pairs from these inconsistent chunks is very error-prone.

Finding true probabilities

After constructing the initial guesses, we use an iterative probabilistic aligner to get the true conditional probabilities for the corresponding grapheme and phoneme sequences. To filter out incorrect guesses and to aid the discovery of new alignments, every conditional probability $P(phonemes, graphemes)$ is taken to be at least $p_{min}^{1.1N_{graphemes}-0.1+0.3|N_{graphemes}-N_{phonemes}|}$, where $N_{graphemes}$ and $N_{phonemes}$ is the number of graphemes and phonemes in the corresponding sequences respectively. Using this

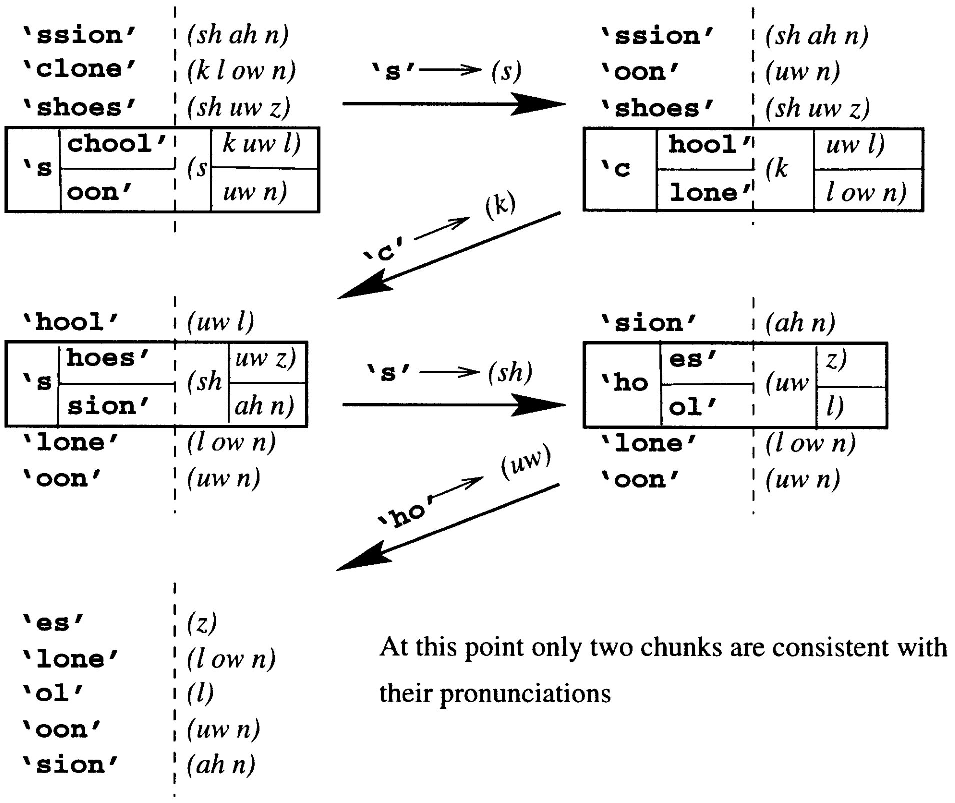

Figure 4-1: Errors arise within the slice algorithm.

probability function forces the aligner to associate a large number of graphemes to an equal number of phonemes.

Aligning phonemes to graphemes

During the last step of the preprocessing, the aligner breaks down multi-phoneme to multi-grapheme alignments into a series of one-to-one alignments. If the alignment had more than one phoneme in it, they are aligned along the graphemes using a dynamic aligner. Individual phonemes are aligned to the grapheme with which they have the highest conditional phoneme-grapheme probability. If there is a tie, the phoneme is assigned to the leftmost grapheme in the tie.

4.1.2 Language-specific phoneme information

The aligner does not need any language-specific information about the phonemes or graphemes, since it works purely based on the conditional probabilities. Therefore, the entire preprocessing can be done without any language-specific information. We discussed, however, in Section 3.3 that phonetic information greatly improves the performance of rule finding, as proposed rules will be more accurate if the scoring function involves some phonetic knowledge.

In our experiments, the precision of the system generally improved with the amount of phonetic knowledge in the system. Therefore, it is advised that an expert describe some language-specific phonetic information. To make this possible, we added a phonetic information structure to our learner. It basically allows an expert to specify:

- the stress marks in a language,
- the vowels of the language (defined as phonemes that need to have stress information),
- phonetic features for some or all phonemes,
- frequent grapheme or grapheme sequence for each phoneme,

- the relative importance of features,
- phonemes that should be treated as equivalent[1], and
- phoneme name substitutions[1].

Nevertheless, asking for phonetic information sacrifices the attempt that the learner be completely automatic and require minimal side information. Fortunately, some information can be extracted purely from the pronunciation dictionaries. We examined the feasibility of some of the techniques to extract these features.

Phoneme-grapheme correspondences

We can extract phoneme-grapheme correspondences from the dictionary using the slice algorithm, described in Section 3.1.1. The method requires linear time $(O(\sum_i length(word_i)))$.

Vowel-consonant qualities

If the pronunciation dictionary contains stress information, we can separate vowels from consonants based on the fact that there should be only one vowel in each syllable, or in other words, for each stress mark. Stress marks are conventionally placed in front of the syllable they refer to, but their exact location is uncertain except for the requirement that they occur before the vowel of the syllable. Therefore, we can calculate the conditional probabilities $P(a \text{ is vowel}|b \text{ is vowel})$ and $P(a \text{ is consonant}|b \text{ is consonant})$. Using these probabilities, we can find which assignment of consonant/vowel qualities is the most feasible. This process should take $(O(\sum_i length(word_i) + N^2_{phonemes}))$ execution time, where $N_{phonemes}$ is the number of different phonemes in the language.

We decided to use phonetic information specified by an expert, as it required negligible effort; thus, the performance gain from the use of this information outweighed

[1]We included phoneme renaming to allow experiments with different numbers of stress levels, and also because some of our old pronunciation systems used different symbols for the same phonemes. This renaming also simplifies comparing other systems' performance to our system.

our desire to keep language-specific information to the bare minimum.

4.1.3 Setup

Before the learning process can start, the learner must be initialized by proposing an initial pronunciation for each word in the training set.

Initial state

The initial pronunciations are calculated based on the individual phoneme-grapheme conditional probabilities. To get the initial pronunciation for each word, we assign the most common phoneme sequence to each grapheme in the word. However, in some cases the most common phoneme sequence is an empty sequence. For example, the most probable transcription of `'e'` in English is an empty sequence, as word-final `'e'`-s in the language are almost always silent. We ran some experiments (described in Section 5.1.3 on page 67), and it turned out to be more advantageous to always assign a non-empty phoneme sequence to a letter. Therefore, if the most probable phoneme sequence for a letter is empty, we assign the second most probable phoneme sequence to that letter.

4.1.4 The learning process

State

In the learning process, we try to find transformations that bring our proposed pronunciations closer to the truth. In Section 3.2 we discussed that it is important to keep phonemes aligned with graphemes individually. To achieve this, we used the representation in Figure 4-2 for the learning process.

Each phoneme in the pronunciation guess is assigned to a letter. Initially, phonemes are aligned with the letter for which they were generated. (This is the *static alignment.*) The pronunciation guess is aligned with the truth, as well, with the necessary corrections marked. Finally, the truth is aligned with the spelling, the same way it was aligned in the training set. This information is used later when the

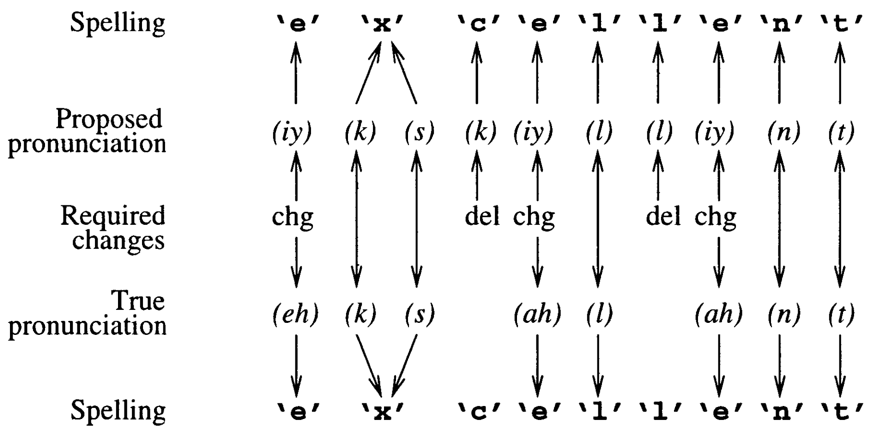
Spelling
'e' 'x' 'c' 'e' 'l' 'l' 'e' 'n' 't'
Proposed pronunciation
(iy) (k) (s) (k) (iy) (l) (l) (iy) (n) (t)
Required changes
chg del chg del chg
True pronunciation
(eh) (k) (s) (ah) (l) (ah) (n) (t)
Spelling
'e' 'x' 'c' 'e' 'l' 'l' 'e' 'n' 't'

Figure 4-2: The phoneme-grapheme alignment scheme used by the learner.

proposed pronunciation approaches the truth, so that we get a more precise notion of which letters the phonemes correspond to in the proposed pronunciation. (We call this process *dynamic alignment.*) We examined the usefulness of this information, and found that it improves the effectiveness of the learned rules.

Finding rules

During the learning process, we look for rules that bring the proposed pronunciations closer to the truth. We then apply one of the rules found to get new pronunciation guesses.

We use a data-driven approach to find useful rules. This means that we compare the proposed pronunciations to the truth, and propose rules at each place where the pronunciations deviate from each other to correct the mistake. The comparison is done by aligning the proposed solution with the truth to find necessary phoneme insertions, deletions or substitutions.

Rules are chosen from a set of allowable rule templates by filling out the blanks. The rule templates we use in the system are:

Change p_i to p_j	if it is aligned with l_i, and is in a certain context C,
Delete p_i	if it is aligned with l_i, and is in a certain context C, and
Insert p_i	in context C.

In these templates, context C can be ‘$\_p_1p_2\ldots p_N$’, ‘$p_1\_p_2\ldots p_N$’, $\ldots$, ‘$p_1p_2\ldots\_p_N$’, or *(_ l_1 l_2 $\ldots$ l_N)*, *(l_1 _ l_2 $\ldots$ l_N)*, $\ldots$, *(l_1 l_2 $\ldots$ l_N _)*, where N denotes the context length and ranges from one to the maximum context length *($1\ldots N_{context}$)*. For example, if the pronunciation guess is

$$\begin{tikzcd}
\text{Word} & \text{`c'} & \text{`h'} & \text{`a'} & \text{`r'} & \text{`m'} \\
\text{Proposed pronunciation} & (ch) \arrow[u] \arrow[d, leftrightarrow] & & ('er) \arrow[u] \arrow[d, leftrightarrow, "chg"] & (r) \arrow[u] \arrow[d, leftrightarrow] & (m) \arrow[u] \arrow[d, leftrightarrow] \\
\text{True pronunciation} & (ch) & & ('aa) & (r) & (m),
\end{tikzcd}$$

,

the proposed rules for maximum context of 3 are

1. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`h_`’
2. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`_r`’
3. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`ch_`’
4. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`h_r`’
5. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`_rm`’
6. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`$ch_`’
7. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`ch_r`’
8. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`h_rm`’
9. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context ‘`_rm$`’
10. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context *(ch _)*
11. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context *(_ r)*
12. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context *($ ch _)*
13. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context *(ch _ r)*
14. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context *(_ r m)*
15. Change *('er)* to *('aa)* if aligned with ‘`a`’ in context *($ ch _ r)*

16. Change *('er)* to *('aa)* if aligned with '`a`' in context *(ch _ r m)*
17. Change *('er)* to *('aa)* if aligned with '`a`' in context *(_ r m \$)*

Once we have enumerated every mismatch between the correct and the proposed pronunciations, we rank the proposed rules based on how effective they are, that is, how much they improve the score. This can be easily calculated if we keep some statistical information around.

For example, the change in the error for rule 8 above

Change *('er)* to *('aa)* if aligned with '`a`' in context '`h_rm`'

is

$$\begin{aligned}
\Delta error &= \Delta error_{worsen} + \Delta error_{improve} \\
&= weight * (N((er), \texttt{`a'}, (er), \texttt{`h\_rm'}) - N((er), \texttt{`a'}, (aa), \texttt{`h\_rm'})) \\
&= weight * ((N((er), \texttt{`a'}, \texttt{`h\_rm'}) - \sum_{i \neq (er)} N_{proposed}((er), \texttt{`a'}, i, \texttt{`h\_rm'})) \\
&\quad - N_{proposed}((er), \texttt{`a'}, (aa), \texttt{`h\_rm'})) \\
&\leq weight * ((N((er), \texttt{`a'}, \texttt{`h\_rm'}) - N_{proposed}((er), \texttt{`a'}, (aa), \texttt{`h\_rm'})) \\
&\quad - N_{proposed}((er), \texttt{`a'}, (aa), \texttt{`h\_rm'})) \\
&= weight * (N((er), \texttt{`a'}, \texttt{`h\_rm'}) - 2N_{proposed}((er), \texttt{`a'}, (aa), \texttt{`h\_rm'}))
\end{aligned} \tag{4.1}$$

where $N(phon, letter, phon_{target}, context)$ is the number of occurrences that *phon* that should be $phon_{target}$ is aligned with *letter* in *context*, $N(phon, letter, context)$ is the number of occurrences that *phon* is aligned with *letter* in *context*, and $N_{proposed}(phon, letter, phon_{target}, context)$ is the number of times the rule "change *phon* to $phon_{target}$ if it is aligned with *letter* in context *context*" was proposed. $Weight$ is a scalar that denotes the significance of the transformation, and it is there because not all editing transformation types (substitution, stress change, deletion, or insertion) are equally significant in the scoring function. A lower bound on the change in the error in (4.1) is

$$\Delta error > -weight * N_{proposed}((er), \text{`}\mathtt{a}\text{'}, (aa), \text{`}\mathtt{h\_rm}\text{'}). \qquad (4.2)$$

As seen in equation (4.1), we need to keep track of only the contextual frequencies and the number of time each rule was proposed. While this is a large amount of information (about 100-500 MBytes for 3000 words with 3 context length), it does significantly speed up the rule search.

At each step of the learning process we select the rule that improves the score the most. If there are several rules that cause the same improvement, we choose the one that is more general (uses smaller context, uses phoneme context as opposed to grapheme context, etc.). Also, we do not apply a rule if it was applied once before because doing so could cause infinite loops if the rule gets re-proposed. To understand why this could happen, it is important to see that when we evaluate the effectiveness of a rule, we do it in the context of the current alignment. Once the rule is applied, the words get rescored; thus, the alignments might change.

4.1.5 Scoring

The scoring of the proposed pronunciations is done using a dynamic aligner that aligns each proposed pronunciation with the true pronunciation. The complexity of the aligner is $O(N^4)$ where N is the number of phonemes in the pronunciation (more precisely it is $O(N^2_{proposed} N^2_{correct})$, where $N_{proposed}$ and $N_{correct}$ are the lengths of the proposed and the correct pronunciations respectively).

The error between the pronunciations is the weighed string edit distance:

$$Error = w_{delete} * N_{delete} + w_{insert} * N_{insert} + w_{sub} * N_{sub} + w_{stress} * N_{stress}, \qquad (4.3)$$

where N_{delete}, N_{insert}, N_{sub}, and N_{stress} are the number of phoneme deletions, insertions, substitution, and stress changes required respectively, and w_{delete}, w_{insert}, w_{sub}, and w_{stress} are the corresponding penalties for each mistake.

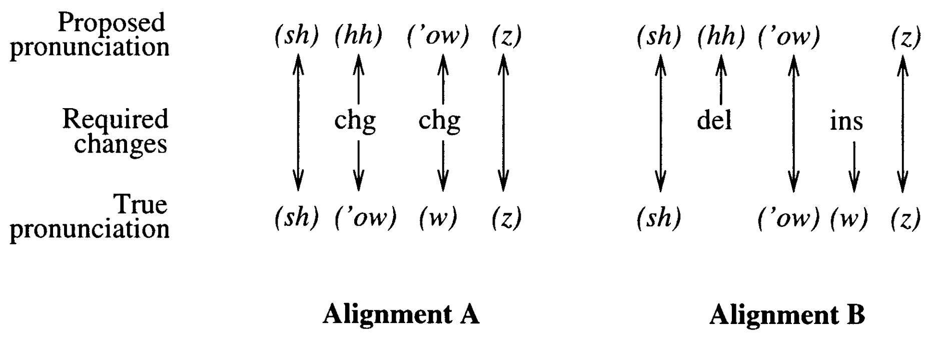

Figure 4-3: Various alignments between *(sh hh 'ow z)* and *(sh 'ow w z)* with the same score.

The rule inducer uses the alignment achieved during the scoring process to propose corrective transformations; therefore, it is important that the proposed and true pronunciations are aligned intelligently, especially if the difference is large. In our system, we weigh insertions, deletions, and substitutions the same way; therefore, it is possible that there are several alignments with equal error. Consider the alignments between *(sh hh 'ow z)* and *(sh 'ow w z)* in Figure 4-3.

The proposed rules for alignment B,

Delete *(hh)* (aligned with l_1) in context *(sh __ 'ow)*.

Insert *(w)* in context *('ow __ z)*

seem more viable than the rules proposed for alignment A:

Change *(hh)* (aligned with l_1) to *('ow)* in context *(sh __ 'ow)*.

Change *('ow)* (aligned with l_2) to *(w)* in context *(hh __ z)*.

Therefore, we augmented the scoring function for the dynamic alignment process with weighed phonetic distances. We assigned weights to each feature we deemed important, and added the distances in feature space as an error to the phoneme substitution penalties. Nevertheless, this additional score was weighed significantly less than the edit distance. Moreover, to speed up distance calculations, we used the negative dot product of the individual feature vectors instead of the absolute distance. As expected, adding this corrective term has significantly improved the overall score of the learner, since the proposed rules became more uniform; thus, the system found the more significant rules earlier.

4.1.6 Applying rules

Rules are applied in a straightforward manner. For substitutions and deletions we check if there is a phoneme that satisfies the contextual requirements (is aligned to the specified grapheme and has the specified phoneme or grapheme context). If there is, the phoneme is simply changed for substitutions, or removed for deletions.

If the transformation is an insertion, we check if the specific context is in the word, and if it is, we insert the phoneme. Insertion is a more complicated procedure because we have to maintain correspondence between graphemes and phonemes throughout the learning process. For insertions this means that we have to instantiate a reasonable correspondence between the newly inserted phoneme and a grapheme. If the context was a grapheme context, the inserted phoneme will be assigned to the grapheme on the right, except when the insertion was at the end of the word. In that case the phoneme is assigned to the last letter. If the context was a phoneme context, we assign the phoneme to the grapheme which is halfway between the graphemes assigned to the neighboring phonemes. If the insertion is at one of the ends of the word, the phoneme is assigned to the grapheme on the edge. The insertion process is illustrated by Figure 4-4.

Rule application is done from left to right. This means that if a rule can apply at several places in a word, it will be first fully applied at the leftmost position. This could potentially result in infinite application of a rule, if the rule creates a new context to the right, where it could apply again. These loops are detected and avoided by the system, so that such rules will only execute once in a single context. Nevertheless, it is possible that they can apply at two or more separate contexts in the same word (Figure 4-5).

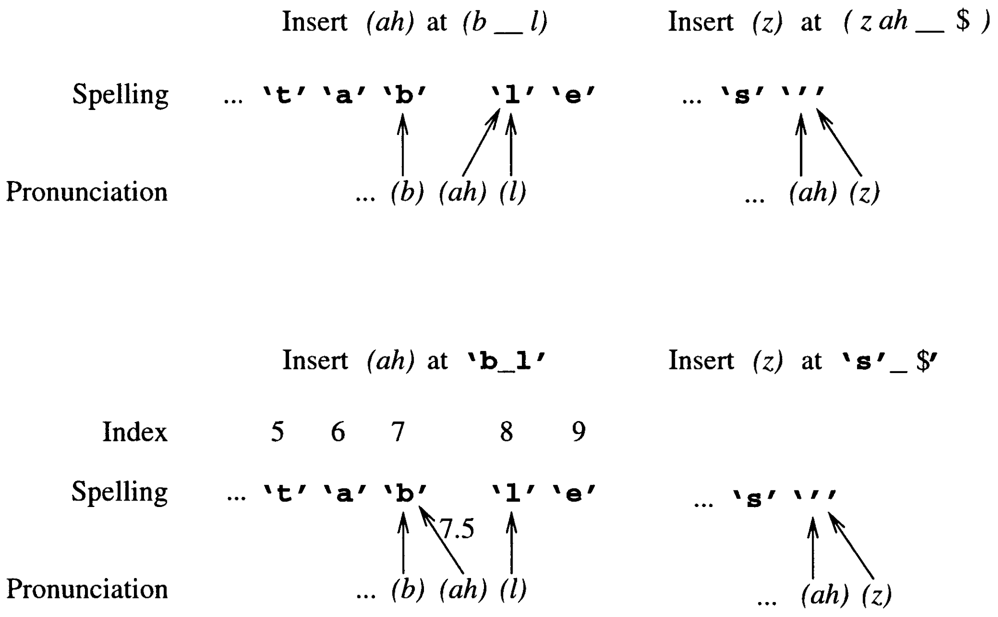

Figure 4-4: Alignments during the phoneme insertion process.

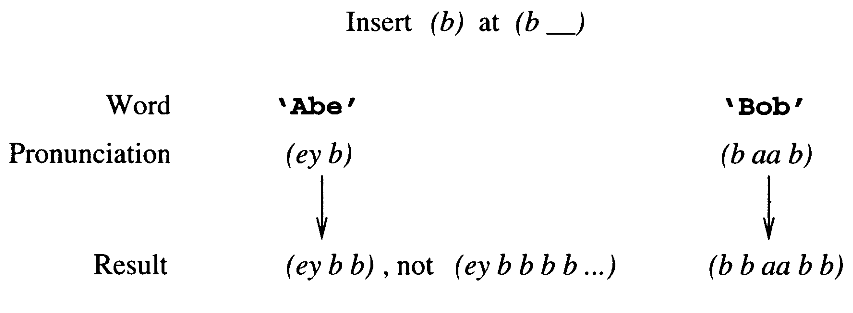

Figure 4-5: Problematic insertions that would result in an infinite loop.

4.1.7 Optimizations

We incorporated several optimization techniques to speed up rule finding and rule application. Rule finding is based on the alignment information produced during the scoring process. We use a dynamic aligner — the fastest available method — for the scoring.

We rank the proposed transformations based on their usefulness, which is calculated from contextual and rule frequencies. To quickly access rules and contextual information, they are stored in a hash table. Rules, furthermore, are also accessible in the order of decreasing frequency (number of times they were proposed), so that when we compare the effectiveness of rules, we can ignore rules that cannot potentially be more effective than the 'current winner' (based on equation (4.2) on page 51). With these optimizations, evaluating a potential rule can be done in constant time.

Also, it is not necessary to rescore and recompute contextual frequencies every time a new rule is to be found. Since rules apply only to a limited number of words, it is sufficient to rescore only the words that were affected by the previous rule. This has significant effects on the run-time performance, since late rules tend to apply to only a couple of words. The contextual frequencies can also be updated incrementally, since they only change at words where the previous rule was applied.

Rule application can also be sped up by storing which context appears at which words, so that when we are looking for a context to apply a rule, we only need to look at the words that actually have that context. This has a significant effect on the speed of rule applications, since late rules only apply to a couple of words, and thus, there is no need to check all the words for the rule context.

4.1.8 Information to be saved

While the learning process requires a large amount of information, the pronunciation rules are captured by a small fraction of that information. The only information we need to store are the

- the initial phoneme sequences for the letters, and

- the transformational rules.

Nevertheless, for debugging and recovery purposes we saved the state of the system at every step, and also saved some additional information about each rule (where it applied, why it was proposed, and how it improved the score) for the purposes of debugging and reasoning.

One important detail is that the phoneme names in the learner are different from the names used by the pronunciation generator because of phoneme renaming. Also, the phoneme extensions mentioned in Section 4.3.2 are encoded differently on the rule-application side because the rule applier does not have all the information that the learner has.

4.2 Generating word pronunciations

Word pronunciations are generated in a straightforward way. At first, an initial pronunciation is produced by transcribing each letter. Then, the transformation rules are applied one after the other. The resulting pronunciation is proposed as the pronunciation for the word.

4.2.1 Information needed

The information that is needed for generating word pronunciations are

- the initial phoneme sequences for the letters, and
- the transformational rules.

This information (referred to as *pronunciation information*) is saved by the learner in a format convenient for the rule applier.

4.2.2 Setup

Before rule application begins, the generator produces a preliminary pronunciation by concatenating the phoneme sequences corresponding to each letter in the word.

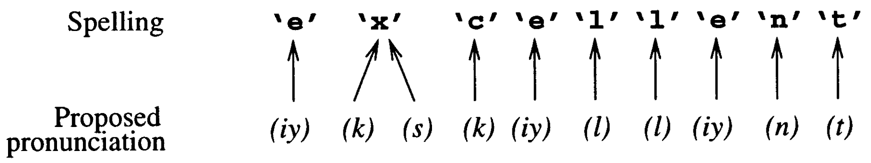
Spelling
'e' 'x' 'c' 'e' 'l' 'l' 'e' 'n' 't'
Proposed pronunciation
(iy) (k) (s) (k) (iy) (l) (l) (iy) (n) (t)

Figure 4-6: The alignment representation used by the rule applier.

There are some potential issues, such as

- what to do with upper/lowercase letters, and
- what to do with letters not in the pronunciation information.

The problem with the case of the letters is resolved by introducing a case sensitivity flag in the rule applier. If it is on, the letters are case sensitive. If it is off, the pronunciation generator first tries to find a transcription for the given case of the letter, and if it fails, it tries the opposite case. This way, it is still possible to give case sensitive initial transcriptions for letters.

We decided to simply produce no transcription for letters not specified by the pronunciation information. We could have removed these letters from the word as well, but that might have some disadvantages. We based our decision on the expectation that the current system will not need to handle characters that are not transcribed by the pronunciation information.

The data structure used during the rule-application process is similar to the one used during learning, except that it has only two layers (Figure 4-6). Phonemes are initially assigned to the letters that instantiated them.

4.2.3 Applying rules

Rule application is equivalent to the rule application during the learning process. The pronunciation generator checks whether the rule applies to the current pronunciation guess. If it does (the rule context is present), the transformation is performed. Deletions and substitutions are straightforward, while insertions follow the same guidelines we outlined in Section 4.1.6. Rules again are applied from left to right, and infinitely applicable rules are detected and applied only once in a single context.

Optimizations

There is a simple optimization performed during the rule-application process. In order to eliminate checking the context for a rule that cannot potentially apply in a word, we check if the word has all the letters that are referenced in the rule. A letter bitmask is calculated for each rule and the word, and a rule can only apply if the bitmask for the word completely contains the bitmask of the rule. Unfortunately, this approach only produced a three-times performance gain.

Further optimizations would have been possible by converting the rules into a finite state transducer, but the implications on the memory requirements prevented us from doing so. With more memory we could have also sped up context matching in a word, but we wanted to have a system that uses a very small amount (less than 50K) of memory.

4.2.4 Generating the pronunciations

Once the generator has tried to apply every rule, it concatenates the phonemes in the proposed pronunciation into a single pronunciation string. Although pronunciations are encoded as directed graphs in the recognizer, the time it took to parse the strings and produce these directed graphs was negligible. We also tried to produce the pronunciation graphs while we concatenated the phonemes, but we experienced no performance gain and a slightly higher memory usage. Therefore, we abandoned this idea.

4.3 Improvements

There are several improvements that we considered for the system. First, the system — as it is — can only produce a single pronunciation for a word, while we wanted to produce multiple pronunciations.

Multiple pronunciations are represented with an acyclic directed graph; thus, the straightforward approach would be to use graph transformations as rule templates. Unfortunately, it is difficult to score proposed graphs in an intelligent way. The problem is not that it is hard to figure out how well the proposed pronunciations match the correct pronunciation(s), but that it is hard to score the proposed graphs by aligning them with the correct pronunciation graphs. We need to do this alignment because the rule proposition is data-driven, and it is important that the scoring function can suggest rules that bring the pronunciations closer to the truth. Finding such a scoring function is not straightforward, especially considering the fact that several pronunciation graphs can encode the same pronunciations.

4.3.1 Compound phonemes

To get closer to our goal, we examined our word pronunciations and discovered that multiple pronunciations in many cases arise simply by having phoneme variants in the pronunciation, such as

`'more'` $\longrightarrow$ *(' m ow (r | . er))*, or

`'hold'` $\longrightarrow$ *(' h ow [l] d)*.

These variants (referred to as *compound phonemes* in the rest of the document) encode either an optional phoneme, or a choice between two possible phonemes. If we encode these variants as phonemes themselves, we can use our current learner without any modification.

To illustrate the importance of using compound phonemes, we categorized the words in our English pronunciation dictionary, and found that only 58.3% of the words have one pronunciation. There are compound phonemes in 21.3% of the corpus, and for a large percentage of them (13.9% of the corpus) the resulting pronunciation after encoding becomes a single pronunciation. Therefore, we can use 72.2% of the English pronunciation corpus using compound phonemes in our pronunciation learner. Unfortunately, compound phonemes cannot simplify 20.4% of the English pronunciation corpus, mostly because the pronunciations vary in more than one phoneme at a time, e.g.

`'abacus'` $\longrightarrow$ *(' ae b . ah | . ah b ' ae) k .ah s)*,

or includes varying stress, such as

`'greenville'` $\longrightarrow$ *(g r ' iy n v (, ih | . ah) l)*.

In 4.4% of the corpus the difference is only the stress of a syllable, e.g.

`'marks up'` $\longrightarrow$ *((' | .) m aa r k s ' ah p)*.

With a script we found and transcribed the compound phonemes into special symbols and using phoneme renaming, we renamed them into their original form for the pronunciation generator. We could even assign feature values for the compound phonemes. The feature set of such phonemes were composed of the common features of the individual phoneme components.

Unfortunately, this solution still cannot handle variants with more than two phonemes, or variants involving phoneme sequences. Another problem is that the stress information is included in the phonemes, and therefore, compound phonemes involving vowels with different stress values *(, ow | ' ow)* or *(, er | . er)* cannot be combined.

Still, using compound phonemes allowed us to train on more words. In fact, without compound phonemes, we completely missed examples for many important pronunciation phenomena; for example, we missed all examples of words ending in `'re'`, such as `'more'`, `'dare'`, etc.

4.3.2 Features

We also investigated the use of features to generalize pronunciation rules. Transformation rules in phonology textbooks often use phonetic features for context description, as in the rule

$$\text{Change } (s) \text{ to } (z) \text{ in context } \left(\begin{bmatrix} \textit{voiced} \\ \textit{consonant} \end{bmatrix} \text{--} \right) \tag{4.4}$$

If we extended our notion of phonemes to include feature bundles, which could potentially stand for multiple phonemes, we could use our learner to discover such general rules. To examine the possibilities, we introduced new pseudo-phonemes described by their feature set. These feature-phonemes (also referred to as feature-groups) can map to any phoneme that matches all of their features. To allow the use of feature-phonemes we needed to modify our system at two places.

When the learner proposes rules to correct mistakes, the rules are expanded so that all equivalent rules using feature-phonemes are also proposed. To do this, we need to substitute the phonemes in the rule with feature groups that can stand for the given phoneme. We need to do these substitutions in all possible ways, and propose each resulting rule.

Suppose, for example, that we only have two features +voice/-voice (+v/-v) and vowel/consonant (V/C). If the following rule is proposed

Change *(s)* to *(z)* in context *(d --)*,

these rules would also be proposed due to the rule expansion

1. Change *([C])* to *(z)* in context *(d --)*
2. Change *([-v])* to *(z)* in context *(d --)*
3. Change *([-v C])* to *(z)* in context *(d --)*
4. Change *(s)* to *(z)* in context *([+v] --)*
5. Change *([C])* to *(z)* in context *([+v] --)*
6. Change *([-v])* to *(z)* in context *([+v] --)*
7. Change *([-v C])* to *(z)* in context *([+v] --)*
8. Change *(s)* to *(z)* in context *([C] --)*

9. Change *([C])* to *(z)* in context *([C] __)*
10. Change *([-v])* to *(z)* in context *([C] __)*
11. Change *([-v C])* to *(z)* in context *([C] __)*
12. Change *(s)* to *(z)* in context *([+v C] __)*
13. Change *([C])* to *(z)* in context *([+v C] __)*
14. Change *([-v])* to *(z)* in context *([+v C] __)*
15. Change *([-v C])* to *(z)* in context *([+v C] __)*

Note that rule 12 is the same as rule (4.4). Nevertheless, this approach still cannot represent phonetic rules, like

$$\begin{bmatrix} -voice \\ consonant \end{bmatrix} \longrightarrow [+voice] \Bigg| \begin{bmatrix} +voice \\ consonant \end{bmatrix} \text{--} \tag{4.5}$$

Another disadvantage of this approach is that the memory requirements for keeping the contextual frequencies explode with the number of features.

$$O(Mem_{required}) = O(\sum_i length(word_i) * (2^{N_{features}})^{length_{context}}) \tag{4.6}$$

A similar explosion happens with the execution time. Due to these problems, it would be preferable if rules could be generalized as a post-processing step, where we combined several rules into one general observation. Unfortunately, combining rules can potentially interfere with subsequent rules, and is therefore risky. We could, nevertheless, store which rules apply to which words and use this information to estimate the effect of a rule generalization.

Another approach would be to examine possible rule generalizations during the learning process, and if a plausible generalization comes up, we could revert the learner to the state where a general rule could be applied, and continue learning from that point. Due to time constraints, we did not investigate these approaches; nevertheless, they are certainly worth looking at in the future.

Chapter 5

Discussion

5.1 Results

We evaluated the performance of our text-to-pronunciation system in two steps. First, we examined the contributions of various system parameters. Then, we compared the fine-tuned system's performance to two existing text-to-pronunciation systems.

We measured the performance of the system using English word pronunciations. During the test runs, we trained the learner on various training sets, with different configurations. We used a separate test set to evaluate the performance of the system. Since the system could generate multiple pronunciations, we measured the following quantities:

Phoneme error rates:

- minimum error: the difference between the closest pronunciations among the correct and generated pronunciations.
- average error: the average difference between the individual pronunciations of the correct and generated pronunciations.

Word error rates:

- exact generation: the number of words for which the generated pronunciation(s) exactly matched the correct pronunciation(s).

- maximal word accuracy: the number of words for which at least one generated word was correct.
- average word accuracy: average percentage of correct word pronunciations.
- undergeneration: the number of words for which there were some correct pronunciations that were not generated by the system.
- overgeneration: the number of words for which the system generated pronunciations that were incorrect. The compliment of this quantity is the measure that all pronunciations generated were correct, which — for words with one generated pronunciations — equals to the word accuracy.

During the evaluation of the system parameters, we also measured

- the number of rules generated,
- the phoneme and word accuracy for the training set,
- the average number of generated pronunciations per word, and
- the percentage of words that at least one rule was applied to (*application percentage*).

5.1.1 Effects of different system parameters on the performance

For the general evaluation of our learning method, and to fine-tune our system, we examined the effects of various system parameters on the learning performance. In particular, we examined the effects of case sensitivity, initial guesses, dynamic versus static alignments during learning, context size, amount of phonetic information in the phoneme-phoneme alignment, and the use of features and compound phonemes.

We used a 1000-word, randomly selected training set containing either words with one pronunciation, or words having only one pronunciation using compound phonemes. The test set was the same for all of the test cases, and it contained 5000

Case sensitivity		Training set		Test set		
Initial guesses	Rule application	Word accuracy	Phoneme accuracy	Exact words	Word accuracy	Phoneme accuracy
No	No	49.4	86.5	21.3	26.3	76.9
No	Yes	N/A	N/A	21.1	26.3	77.0
Yes	No	N/A	N/A	21.1	26.1	76.8
Yes	Yes	49.5	86.6	21.1	26.1	76.7

Table 5.1: Effects of case sensitivity during rule application and initial guesses.

randomly selected words. The training and test sets had no common elements. We also ignored stress for the evaluation; nonetheless, we separated schwas from the rest of the vowels using separate phoneme names. The results for the different test runs are summarized in Table A.3.

5.1.2 Effects of case sensitivity

The learner is capable of deriving pronunciation rules both case-sensitively or case-insensitively. The main reason for this option was that in many languages capitalization does not influence the pronunciation of words. However, some languages can rely on case for pronunciation; therefore, we did not want to rule out the importance of case.

The rule applier can also treat pronunciation rules or initial pronunciation guesses case sensitively. As mentioned in Section 4.2.2, it is possible to give initial pronunciations for letters of either case. If the initial guesses are case insensitive, the rule applier will look at each letter, and for letters that do not have an initial pronunciation associated in their proper case, it will use the initial pronunciation for the opposite case. Case sensitivity during rule application means that the graphemes in the rule contexts and the words are matched case sensitively.

As seen in Table 5.1, case sensitivity has minimal influence on the learning performance in English. Nevertheless, it is shown that it is not advised to use case sensitive initial guesses. On the contrary, case sensitivity during rule application slightly improves phoneme accuracy, although it lowers the number of exact word

		Training set			Test set		
Initial guesses	# of rules	Word accur.	Phon. accur.	Application percentage	Exact words	Word accur.	Phon. accur.
Can be empty	449	85.3	43.8	94.1	16.5	20.0	73.9
Non-empty	337	86.5	49.4	96.1	21.3	26.3	76.9

Table 5.2: Effects of allowed empty initial guesses.

pronunciations.

5.1.3 Effects of allowed empty initial phoneme sequences

We also examined the effect of allowing or disallowing empty initial pronunciation sequences in the learning system. In some cases, empty sequences are more probable in phoneme-grapheme alignments than non-empty ones. For example, in English the following letters have highly probable empty transcriptions:

Grapheme	Empty sequence probability	Best non-empty phoneme sequence	Probability
' '	1.00	∅	N/A
'''	0.97	*(.ah)*	0.02
'e'	0.57	*('eh)*	0.13
'g'	0.52	*(g)*	0.32
'h'	0.71	*(hh)*	0.21
'o'	0.20	*(.ah)*	0.16
'u'	0.35	*('ah)*	0.30

As shown on Table 5.2, allowing empty initial sequences results in more rules. This is because the learner needs to insert the omitted phonemes into the words. In fact, the learner has generated rules to insert the missing phonemes fairly early:

8. Insert *(hh)* in context '$_h'

15. Insert *(g)* in context '$g_'

In addition, even though more rules are generated, they apply to fewer words in the rule application process. The performance of the system significantly dropped when we allowed empty initial phoneme sequences. Therefore, we disallowed empty initial sequences in our learning processes.

	Test set		
Grapheme-phoneme alignment	Exact words	Word accuracy	Phoneme accuracy
Initial, then true alignment	21.3	26.3	76.9
Always true alignment	21.3	26.4	77.1

Table 5.3: Effects of different alignment schemes during learning.

5.1.4 Dynamic versus static alignment during the learning process

In Section 4.1.4 (under *State*) we discussed that the true pronunciations are aligned with the spelling during the learning process. When the proposed pronunciation approaches the truth, we use these true alignments — as opposed to the initial alignments — to match phonemes with graphemes. We examined how the system performance changes if we use the true alignments from the start.

As seen in Table 5.3, the system's performance slightly improves if we use the true alignments from the beginning. Therefore, we used true alignments in the subsequent performance tests.

5.1.5 Effects of phonetic information

We also examined how the amount of phonetic information during phoneme-phoneme alignment influences the performance of the learner. We ran several tests with various amounts of phonetic information, including voicing (v), place of articulation (p), vowel height (h), vowel position (f), consonant quality (s: stop, fricative, etc.), roundedness (r), nasal quality (n), and category (c: diphthong, syllabic, etc.) attributes. The vowel/consonant distinction is inherent in the system, so we do not list that information.

The overall result of the tests (Table 5.4) showed that while some information (v, h, f) improves the performance of the learner, certain phonetic information is detrimental to the system (s, n, c, r). To highlight this observation, the system's performance

	Training set		Test set		
Phonetic information	Word accuracy	Phoneme accuracy	Exact words	Word accuracy	Phoneme accuracy
none	42.1	83.8	17.3	21.6	74.0
v, p	49.3	86.6	20.4	25.2	76.3
v, f	50.1	86.9	21.4	26.4	77.0
v, f, h	49.5	86.7	21.5	26.8	77.3
v, f, h, p	52.1	87.8	22.1	27.5	77.8
v, f, h, p, s, n, c, r	49.9	86.6	21.2	26.1	76.9
v, f, h, p, s, r	49.5	86.6	21.3	26.4	77.1
s, c, n, r	42.1	83.8	17.3	21.5	73.9

Table 5.4: Effects of various phoneme information during phoneme-phoneme alignment.

using only these features (s, n, c, r) is worse than when using no information at all.

5.1.6 Effects of compound phonemes

We also examined the performance gain from using compound phonemes. We contrasted the learned pronunciations from a training set of 1000 words with one pronunciation, with ones learned from a training set of 1000 words having only one pronunciation using compound phonemes. Table 5.5 shows the performance of the system with the two training sets. It seems that the performance on the training set decreases if we use compound phonemes. However, it is important to see that contrary to the previous test cases, the words in the two training sets belong to different word categories. Therefore, the decreased performance during the learning stage is

		Training set				Test set		
Phoneme set	# of rules	word acc.	phon. acc.	appl. perc.	aver. pron.	exact words	word acc.	phon. acc.
Regular	381	49.5	86.6	96.1	1	21.3	26.4	77.1
Compound	432	42.4	84.3	96.7	1.06	20.3	25.2	78.1

Table 5.5: Effects of compound phonemes.

not significant. In fact, rules learned with the compound phonemes improved the application percentage, and the phoneme accuracy.

Note, that the word accuracy decreased with compound phonemes. Nevertheless, we must not neglect the fact that compound phonemes allow us to use a wider range of words in the training set, and thus the learner is able to learn and generate a larger set of pronunciation phenomena, including ones observed in some of the most common words.

5.1.7 Effects of context length

As expected, larger context length gives better performance. However, training with large context sizes is prone to overlearning, in which case the system merely memorizes the training set. This happened in our experiment at maximal context size of three (see Table 5.6), which had a lower performance than the run of context size two.

The overlearning is supported by the fact that although there were more 3-context rules than 2-context ones, they applied to fewer words, suggesting that more of the 3-context rules were specific to the training set. Figure 5-1 illustrates the goodness of the rules from runs of various context lengths. It shows the word accuracy achieved by the first N rules learned in the three runs. As seen, after the first 250 rules, 2-context rules always overperform 3-context rules.

Note also, that 1-context rules overperform the higher context ones up until about eighty rules. This illustrates that choosing the most corrective rule does not give optimal performance, as higher-context rules could generate all 1-context rules, but

		Training set			Test set		
Context size	# of rules	word acc.	phon. acc.	appl. perc.	exact words	word acc.	phon. acc.
1	381	49.5	86.6	96.1	21.3	26.4	77.1
2	784	97.7	99.6	98.0	30.8	38.2	82.9
3	792	99.7	100.0	97.7	30.7	38.2	82.8

Table 5.6: Effects of context size on the performance.

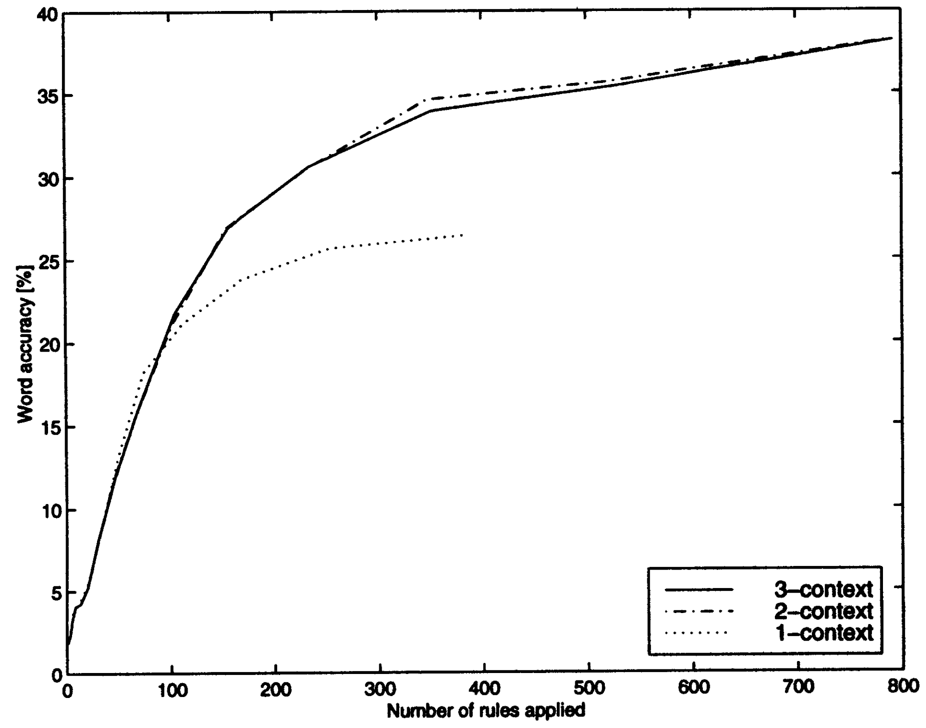

Figure 5-1: The word correctness rate, depending on the context size and the number of rules applied.

still have lower performance in that region.

5.1.8 Effects of feature-phonemes

We also ran some simple experiments using feature-phonemes (Section 4.3.2) of different sets of phonetic features. We generally found that the use of feature-phonemes reduces the number of rules; therefore, the learner does indeed make generalizations based on features. However, the usefulness of features is questionable. While increasing the set of features improved the performance for the 1-context case, it generally decreased the performance for the 2-context case (Table 5.7).

Another interesting detail is that the learned rules applied to the same number of words using features as without them. This could signal that real generalizations did not actually take place. Nevertheless, our experiments with features are preliminary, as the time and memory requirements of the system prevented us to allow a larger

			Training set			Test set		
Context size	Features used	# of rules	word acc.	phon. acc.	appl. perc.	exact words	word acc.	phon. acc.
1	none	381	49.5	86.6	96.1	21.3	26.4	77.1
1	voice	362	49.1	86.5	96.2	21.0	26.7	77.1
1	V/C	375	51.4	87.0	96.1	21.9	27.1	76.9
1	V/C, voice	371	51.4	87.0	96.1	22.2	27.2	76.7
2	none	784	97.7	99.6	98.0	30.8	38.2	82.9
2	V/C	712	97.7	99.6	98.1	30.5	37.7	81.9
2	V/C, voice	697	97.8	99.6	98.0	29.9	37.0	81.0

Table 5.7: Effects of using feature-phonemes.

set of phonetic features. Therefore, a separate investigation is necessary to determine if using feature-phonemes in the transformation-based learner is useful.

5.1.9 Comparison with simple rule-based system

In the second part of our performance evaluation we compared our system's performance to two other systems. As before, we did not differentiate between vowels with different stress levels, but we treated schwas and regular vowels separately. We used a maximal context length of three, and the training set was selected from the words that could be transcribed in a way — using compound phonemes — that they had one pronunciation. The test set was selected randomly among all words in the dictionary, including words that the pronunciation generator would theoretically not be able to generate.

First, we compared our system to `dmakeup`, a context-based system used at Texas Instruments' Speech Recognition Group, which was built similarly to ARPA's approach described in Section 1.3.2 on page 15. Table 5.8 shows the results of the comparison. Our pronunciation learner outperformed the simple context-based system in every case. Note, however, that the number of rules for our system is significantly higher than that for the simple system. This, and the complexity of our rules, results

system	TTP	TTP	TTP	TTP	dmakeup[1]
training set	1K (compound)	2K (compound)	5K (compound)	10K (compound)	N/A
test set	10K (any)	10K (any)	10K (any)	10K (any)	10K (any)
exact generation	33.5 %	44.9 %	62.6 %	73.9 %	19.8 %
under generation	65.8 %	54.0 %	36.7 %	25.6 %	80.2 %
over generation	60.3 %	48.6 %	30.2 %	18.7 %	72.2 %
minimum phon. error	16.9 %	12.7 %	7.5 %	4.5 %	23.6 %
average phon. error	17.1 %	13.1 %	7.7 %	4.7 %	23.6 %
execution time	1.61 ms	2.72 ms	4.18 ms	5.76 ms	100 μ s
number of rules	1061	1721	2890	4082	$\tilde{3}00$

Table 5.8: Performance of the system compared to a simple rule-based system (dmakeup).

in a much higher pronunciation-generation time.

5.1.10 Comparison with neural networks

We also compared our system to Picone's neural network based system [11]. He trained his system on proper names using neural networks with different number of layers. Our system has better performance than the neural network based system on each test set (Table 5.9).

5.2 Evaluation

The transformation-based error-driven learner outperformed both systems that we previously used for generating pronunciations. When evaluating the system, we observed the general tendencies of learning processes. The more information we gave

system	TTP	TTP	TTP	NN	NN	NN
training set	15K (names)	15K (names)	15K (names)	15K (names)	15K (names)	15K (names)
test set	3.5K (names)	3.5K (names)	3.5K (names)	3.5K (names)	3.5K (names)	3.5K (names)
has correct generation	61 %	69.6 %	68.6 %	33.1 %	33.1 %	33.1 %
minimum phoneme error	7.3 %	7.1 %	7.5 %	N/A %	N/A %	N/A %
average phoneme error	8.0 %	7.9 %	8.2 %	N/A %	N/A %	N/A %
number of rules	6594	6667	6619	300 neurons	300 neurons	300 neurons

Table 5.9: Performance of our system compared to a neural network approach.

to the system, the better performance it had. These were the tendencies that we discovered in our system:

- Larger context in the rule templates resulted in better pronunciation generation.
- More phonetic information during learning resulted in rules with better performance.
- Including more cases in the training corpus allowed to learn more pronunciation phenomena.
- Larger training corpus exposed the learner to more examples for special cases.

One of the most important things we learned is that the training set for an automated learner has to be carefully constructed. Throughout the training process we relied on a randomly selected training corpus. Later, when we examined the performance of the system, we realized that many of the radical rules (that caused the system to generate horrendous pronunciations for simple words) resulted because there was no example in the training corpus for the more common pronunciation rules. This happened mainly because we had to restrict our training set to words with one pronunciation, and most common words have a large number of pronunciations.

There are several work-arounds to this problem. We could, for example, randomly include just one of the pronunciations for words that have truly multiple pronunciations. This could possibly result, however, in contradicting examples for competing rules, which could cause the learner to abandon each of the rules involved.

It would also help to have a utility that checks whether the training corpus has all the interesting cases. If the utility found a context that is not represented in the training corpus, it could signal the user so that (s)he can evaluate the importance of the missing example. The utility could also recommend additions to the training set, or the removal of redundant examples.

Another observation was that rules at the late stages of the learning process tend to correct a specific "mistake" that relates to only one word. This means that in the late stage the learner memorizes pronunciations instead of learning pronunciation rules. We think that by the introduction of features this memorization could be greatly reduced. However, we need to find a way to include features without imposing impossible time or memory requirements on the system.

5.3 Future extensions

There are many possible extensions to the system that could resolve problems we noticed during our design and evaluation. Our system also has some deficiencies that need to be corrected:

5.3.1 Training set and Multiple pronunciations

- The system, as it is, cannot truly handle words with multiple pronunciations. However, there is hope that the learning can be adapted to handle directed acyclic graphs representing true multiple pronunciations, since transformation-based error-driven learning was successfully applied to trees [1, 13].

- The training set selection process could use more intelligence as well. In the current system the training set was selected randomly from the pronunciations that the system could handle. A selection process could examine whether all phonetic contexts are represented in the training set and recommend extra training data accordingly. It could also suggest removing redundant elements from the training set. Furthermore, the selection process could be fully automated so that no human intervention is required.

5.3.2 Features and generalization

- Currently the system cannot — in practice — use features during the learning process because of the excessive memory requirements resulting from feature-phonemes. Perhaps the current data-driven approach could be changed to reduce overall space and time requirements without sacrificing the run-time performance. Also, some intelligence for suggesting generalized rules could help, although it could potentially sacrifice language independence.

- Generalization could also be achieved after the learning process. Rules could be collapsed during a post-processing step. Here, care needs to be taken so that the rule generalization does not interfere with the effect of other rules.

- Generalization could also occur during the learning process by recognizing general trends in the proposed rules. For example if several rules with the same semantics are proposed, the learner could collapse them into a single rule. This would require an extension of the current rule selection process.

5.3.3 Performance

- The performance of the system could be improved by using transformation-based error-driven learning as a post-processor to an existing text-to-pronunciation system. We chose not to do this, as our approach requires the alignment of phonemes and graphemes at rule application, which is a time consuming process. However, if the preceding system produced the alignment automatically, our system could be used without significant performance degradation.

- One problem with data-driven rule suggestion is that if our data is not correctly aligned, we can suggest rules that do not make sense or decrease the score. Currently the system has no mechanism of preventing such rule propositions, but it could be augmented to deal with such cases intelligently.

5.3.4 Usefulness

- Currently, the sole purpose of the system is to produce a pronunciation for a word, given it has the pronunciation rules specific to the language. If we have pronunciation rules for several languages, the system could be augmented to be able to pronounce words based on rules for several languages. Using a language identification tool, the system could be extended to propose pronunciations automatically in the correct language. This would be especially useful when trying to recognize proper names, as some people utter names in their own native language, while others try to utter them in the original language of the proper name. One potential problem with this desired feature is that different languages have different phoneme sets, and the pronunciations generated under the rules of one language would need to be translated into the phonemes of the other language.

- We could also use an unsupervised learner to discover pronunciations for languages without knowing the proper pronunciations for any word. If we know how letters can be pronounced, we can use an unsupervised learner [3] to discover pronunciation rules. Probably this would only be useful to discover the general pronunciation rules for a language, as no special cases are marked. An unsupervised learner could also be used as a language identifier.

Appendix A

Tables

[1] 10-300 MB, if optimized for speed
[2] 1-4 MB, mostly probabilistic information (estimate)
[3] 200-700 KB, mostly activation values and data for the backpropagation algorithm (estimate)
[4] human transcription is needed
[5] human experts are needed
[6] although neural network will have different structure
[7] WP: word pronunciations
[8] PR: language-specific pronunciation rules
[9] PhR: language-specific phonological rules
[10] PhGM: basic phoneme-grapheme mappings
[11] PhGA: basic phoneme-grapheme alignments
[12] Morph: morphological analysis
[13] PhQ: phonetic feature description of phonemes
[14] For dictionary only. More, if optimized for speed
[15] if optimized for speed
[16] N/A: Not applicable
[17] N/D: Data not available

Table A.1: Footnotes for Table A.2.

Criteria	Dictionary	Simple Context-based	Rule-based transliterator	Transform. based Error-driven Learner	Parsing Word Morphology	Overlapping Chunks	Neural Networks
Pronunciation Learning							
Space requirements	N/A[16]	N/A[16]	N/A[16]	Huge[1]	Large[2]	N/A[16]	Moderate[3]
Time complexity	Human, months	Human, 3-5 days	Human, 3-5 days	1-5 days	N/D[17]	N/A[16]	N/D[17]
Automatic	No[4]	No[5]	No[5]	Yes	Yes	Yes	Yes
Language Independent	N/A[16]	No	No	Yes	Yes	Yes	Yes[6]
Information needed	WP[7]	PR[8], and PhR[9]	PR[8], and PhR[9]	PhGM[10], WP[7], PhQ[13]	PhGA[11]and Morph[12]	WP[7]	WP[7], PhQ[13]
Pronunciation Genaration							
Memory requirements	Negligible	Tiny (1-2K)	Tiny (1-3K)	Small (5-15K)	Large (100K-1M)	Large (1-3M)[14]	Large (100K-1M)
Disk space requirements	1-2 MB (for dictionary)	1-2 KB	1-3 KB	Small (5-15K)	100K-1M	1-2 MB (for dictionary)	100K-1M
Time complexity	O (1)	$O\ (N_{rules})$	$O\ (N_{rules})$	$O\ (N_{rules})$	$O\ (N_{letters})$	$O\ (N^4_{letters})$[15]	$O(\frac{1}{2}N_{letters}\cdot(N_{phones}+N_{letters}))$
Performance	English	English	English (no proper names)	English	English (Brown corpus)	English (also pseudo-words)	English (surnames)
Phoneme accuracy [%]							
Learning set	100	N/A[16]	N/A[16]	N/D[17]	94.2	N/A[16]	N/D[17]
Test set	0	76.4	90	88	91.7	91.7	N/D[17]
Word accuracy [%]							
Learning set	100	N/A[16]	N/A[16]	N/D[17]	77.3	N/A[16]	54.1
Test set	0	27.8	N/D[17]	N/D[17]	69.3	56.6	44.3

Table A.2: Data sheet of existing TTP systems. Footnotes are listed on Table A.1

Features in alignment[1]	Maximum Context length	Initial guess[2]	Case sensitivity	Feature-phonemes[3]	Alignment	Compound phonemes	Number of rules	Phoneme accuracy (training set)	Word accuracy (training set)	Application percentage	Pronunciations per word	Exact word generation	Undergeneration	Overgeneration	Word accuracy (test set)	Phoneme accuracy (test set)
vfhpsr	1	non-E	no	none	static	none	377	86.5	49.4	96.1	1	21.3	78.7	73.7	26.3	76.9
vfhpsr	1	non-E	yes	none	static	none	381	86.6	49.5	96.0	1	21.1	78.9	73.9	26.1	76.7
vfhpsr	1	non-E	rules	none	static	none	"	"	"	95.9	1	21.1	78.9	73.7	26.3	77.0
vfhpsr	1	non-E	initial	none	static	none	"	"	"	96.0	1	21.1	78.9	73.9	26.1	76.8
vfhpsr	1	empty	no	none	static	none	449	85.3	43.8	94.1	1	16.5	83.5	80.0	20.0	73.9
vfhpsr	1	non-E	no	none	dyn.	none	381	86.6	49.5	96.1	1	21.3	78.7	73.6	26.4	77.1
none	1	non-E	no	none	dyn.	none	497	83.8	42.1	95.6	1	17.3	82.7	78.4	21.6	74.0
s,c,n,r	1	non-E	no	none	dyn.	none	495	83.8	42.1	95.6	1	17.3	82.7	78.5	21.5	73.9
v,f,h,p	1	non-E	no	none	dyn.	none	399	87.8	52.1	97.2	1	22.1	77.9	72.5	27.5	77.8
v,f,h	1	non-E	no	none	dyn.	none	378	86.7	49.5	96.0	1	21.5	78.5	73.2	26.8	77.3
v,p	1	non-E	no	none	dyn.	none	394	86.6	49.3	96.2	1	20.4	79.6	74.8	25.2	76.3
v,f	1	non-E	no	none	dyn.	none	385	86.9	50.1	96.2	1	21.4	78.6	73.6	26.4	77.0
vfhpsncr	1	non-E	no	none	dyn.	none	383	86.6	49.9	96.0	1	21.2	78.8	73.9	26.1	76.9
vfhpsr	1	non-E	no	none	dyn.	yes	432	84.3	42.4	96.7	1.06	20.3	79.3	75.4	25.2	78.1
vfhpsr	1	non-E	no	v	dyn.	none	362	86.5	49.1	96.2	1	21.0	79.0	73.3	26.7	77.1
vfhpsr	1	non-E	no	V/C	dyn.	none	375	87.0	51.4	96.1	1	21.9	78.1	72.9	27.1	76.9
vfhpsr	1	non-E	no	v,V/C	dyn.	none	371	87.0	51.4	96.1	1	22.2	77.8	72.8	27.2	76.7
vfhpsr	2	non-E	no	V/C	dyn.	none	712	99.6	97.7	98.1	1	30.5	69.5	62.3	37.7	81.9
vfhpsr	2	non-E	no	v,V/C	dyn.	none	697	99.6	97.8	98.0	1	29.9	70.1	63.0	37.0	81.0
vfhpsr	2	non-E	no	none	dyn.	none	784	99.6	97.7	98.0	1	30.8	69.2	61.8	38.2	82.9
vfhpsr	3	non-E	no	none	dyn.	none	792	100.0	99.7	97.7	1	30.7	69.3	61.8	38.2	82.8

Table A.3: The system's performance with various system parameters.

[1]Abbreviations are *v* – voicing, *p* – place of articulation, *h* – vowel height, *f* – vowel position, *s* – consonant quality, *r* – roundness, *n* – nasal quality, and *c* – category (diphthong, syllabic, etc.)

[2]Abbreviations are empty – empty initial sequences are allowed, non-E – they are disallowed

[3]Abbreviations are v – ±voice, V – vowel, and C – consonant.

Appendix B

Figures

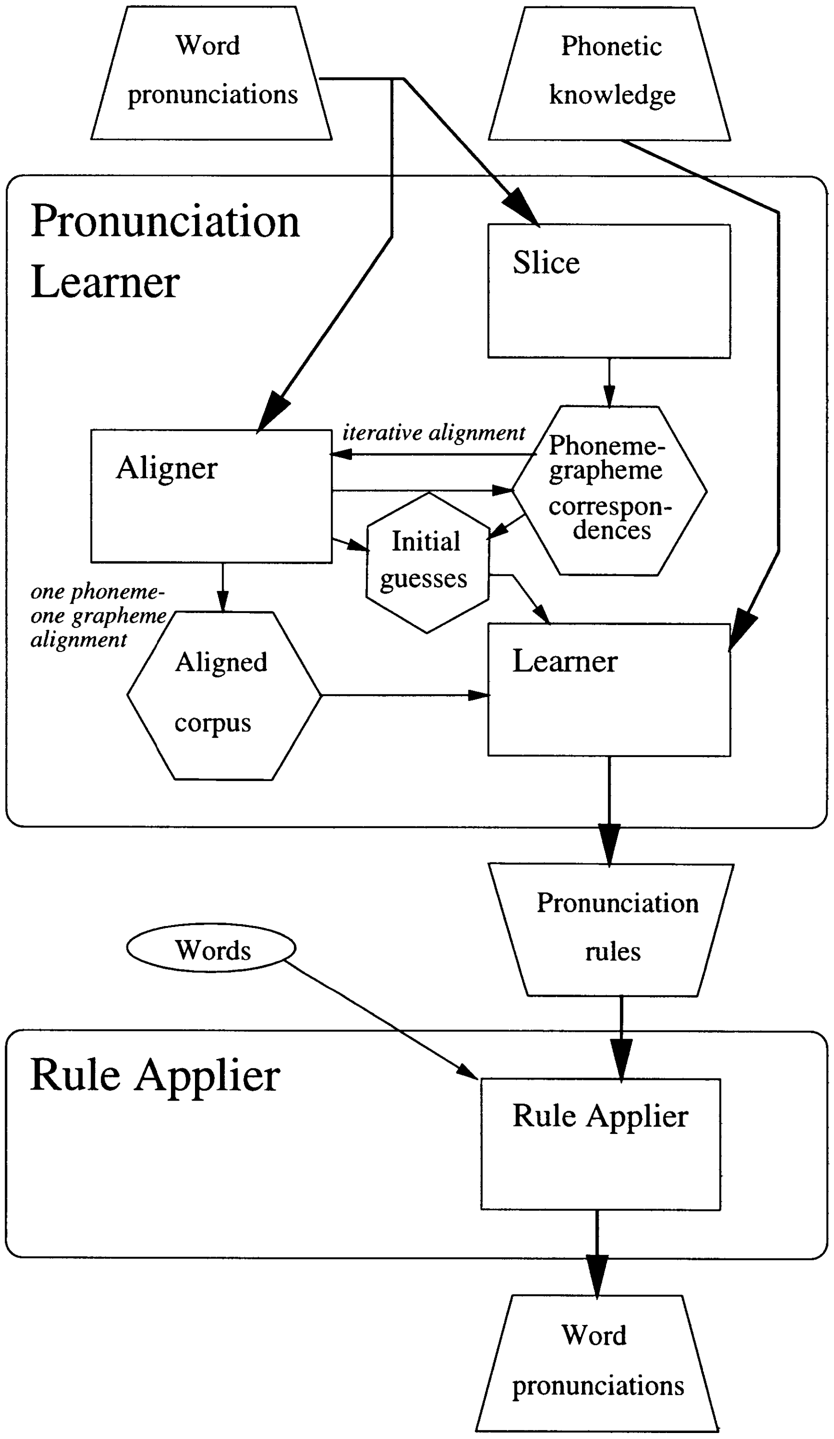

Figure B-1: System overview

Bibliography

[1] Eric Brill. Transformation-based error-driven parsing. In *Association of Computational Linguistics.* ACL, 1993.

[2] Eric Brill. *Transformation-based error-driven learning and natural language processing: a case study in part of speech tagging.* PhD thesis, The Johns Hopkins University, Department of Computer Science, 1995. also in *Computational Linguistics, Dec. 1995.*

[3] Eric Brill. Unsupervised learning of disambiguation rules for part of speech tagging. In *Natural Language Processing Using Very Large Corpora.* Kluwer Academic Press, 1997.

[4] Noah Chomsky and M. Halle. *The sound pattern of English.* Harper & Row, New York, New York, USA, 1968.

[5] Ilyas Cicekli and H. Altay Güvenir. Learning translation rules from a bilingual corpus. In *Proceedings of the International Conference on New Methods in Language Processing*, number 2, Ankara, Turkey, 1996. NeMLaP. also in *http://xxx.lanl.gov/list/cmp-lg/9607#cmp-lg/9607027.*

[6] Walter Daelemans, Peter Berck, and Steven Gillis. Unsupervised discovery of phonological categories through supervised learning of morphological rules. In *Proceedings of International Conference on Computational Linguistics*, Copenhagen, Denmark, 1996. COLING. also in *http://xxx.lanl.gov/list/cmp-lg/9607#cmp-lg/9607013.*

[7] Charles T. Hemphill. *EPHOD, Electronic PHOnetic Dictionary.* Texas Instruments, Dallas, Texas, USA, 1.1 edition, 12 May 1995.

[8] Caroline B. Huang, Mark A. Son-Bell, and David M. Baggett. Generation of pronunciations from orthographies using transformation-based error-driven learning. In *International Conference on Speech and Language Processing (ICSLP)*, pages 411–414, Yokohama, Japan, 1994. ICSLP.

[9] Lidia Mangu and Eric Brill. Automatic rule acquisition for spelling correction. In *International Conference on Machine Learning*, Nashville, Tennessee, USA, 1997. ICML.

[10] Helen Meng, Sheri Hunnicutt, Stephanie Seneff, and Victor Zue. Reversible letter-to-sound/sound-to-letter generation based on parsing word morphology. In *Speech Communtication, Volume 18*, number 1, pages 47–63. North-Holland, 1996.

[11] Joe Picone. Advances in automatic generation of multiple pronunciations for proper nouns. Technical report, Institute for Signal and Information Processing, Mississippi State, Mississippi, USA, 1 September 1997.

[12] Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. Research report cs-tr-532-96, Princeton University, Princeton, New Jersey, USA, 1997. also in *http://xxx.lanl.gov/list/cmp-lg/9610#cmp-lg/9610005.*

[13] Giorgio Satta and Eric Brill. Efficient transformation-based parsing. In *Association of Computational Linguistics.* ACL, 1996.

[14] Helmut Schmid. Part-of-speech tagging with neural networks. In *International Conference on New Methods in Language Processing*, number 1, Machester, England, 1994. NeMLaP. also in *http://xxx.lanl.gov/list/cmp-lg/9410#cmp-lg/9410018.*

[15] Richard Sproat. Multilingual text analysis for text-to-speech synthesis. In *ECAI Workshop on Extended Finite-State Models of Language*. ECAI, 1996. also in *http://xxx.lanl.gov/list/cmp-lg/9608#cmp-lg/9608012*.

[16] Francois Yvon. Grapheme-to-phoneme conversion using multiple unbounded overlapping chunks. In *Proceedings of the International Conference on New Methods in Language Processing*, number 2, Ankara, Turkey, 1996. NeMLaP. also in *http://xxx.lanl.gov/list/cmp-lg/9608#cmp-lg/9608006*.