Learning Pronunciation Rules for English Graphemes Using the Version Space Algorithm

Howard J. Hamilton
Jian Zhang

Technical Report CS-93-02
December, 1993

Department of Computer Science
University of Regina
Regina, Saskatchewan
S4S 0A2

ISSN 0828-3494
ISBN 0-7731-0252-3

Abstract

We describe a technique for learning pronunciation rules based on the Version Space algorithm. In particular, we describe how to learn pronunciation rules for a representative subset of the English graphemes. We present a learning procedure called LEP-G.1 (learning to pronounce English graphemes) that learns English pronunciation rules from examples in the form of word-pronunciation pairs. With our approach, we can translate not only English words in dictionaries, but also new words such as **tuple**, **pixel**, and **deque** which are not found in dictionaries. An experiment where LEP-G.1 learned pronunciation rules for 12 graphemes strongly suggests that learning the other possible 52 graphemes in English is feasible.

1 Introduction

We describe a technique for learning pronunciation rules based on the Version Space algorithm. In particular, we describe how to learn pronunciation rules for a representative subset of the English graphemes. The present work is part of an overall project (LEP-W) to learn how to translate English words to the International Phonetic Alphabet (IPA), a system of symbols representing all individual sound that occurs in any spoken human languages.

The task of learning to translate English words to IPA symbols involves recognizing grapheme(s) of a word, separating the word into syllables, distinguishing open and closed syllables, classifying the **stresses** of each syllable in a word, learning pronunciation rules for graphemes, and accumulating the pronunciation rules that have been learned [Zhang, 1993a]. In this paper, we present a learning procedure called LEP-G.1 (learning to pronounce English graphemes) that learns English pronunciation rules from examples in the form of word-pronunciation pairs. With the pronunciation rules obtained by LEP-G.1, LEP-W can translate not only existing English words in dictionaries but also new words such as **tuple**, **pixel**, and **deque** which are not found in dictionaries. The approach that we have just described is equally applicable to all other phonetic languages. In this paper, we concentrate on learning pronunciation rules for English.

We briefly describe, in the following paragraphs, the overall problem of learning to translate English words to IPA symbols, and then we describe in detail the specific problem, learning pronunciation rules for English graphemes. More detail on the overall approach is given in [Zhang, 1993a].

A *grapheme* is one letter or "the sum of letters and letter combinations that represent a single phoneme" [Morris, 1991]. For example, the graphemes in the word **cat** are **c**, **a**, and **t** and those in **watch** are **w**, **a**, and **tch**. A *phoneme* is the smallest unit of a language which distinguishes meanings. For example, **cat** and **cut** are distinguished by the two individual phonemes /ash/ and /inverted_v/ for the middle phonemes. Graphemes in English may have one, two, or three letters; e.g., **ght** is a three-letter grapheme. Some graphemes represent vowel sounds, while some represent consonant sounds. First, LEP-W will learn how to recognize different types of these graphemes by positive and negative examples.

Secondly, LEP-W will learn how to separate the word into syllables. A *syllable* consists one and only one vowel sound and any number of consonant sounds. LPE-W will learn to recognize the patterns of

consonants and vowels that may constitute a syllable. For example, the word **computer** may be represented by [CVCCVCV] where C stands for a consonant grapheme, and V stands for a vowel grapheme. From a series of examples, each consisting of a word and the syllabicated version of the word, LPE-W will learn that [V] may be a syllable, [CV] may be a syllable, but [CC] may not be a syllable. Gradually, LPE-W will learn the syllabication rules.

The third step is to distinguish open and closed syllables. An *open* syllable ends with a vowel and a *closed* syllable ends with a consonant [Mackay, 1987]. For LPE-W to learn to recognize an open syllable, words ending with vowels will be used as positive examples and words ending with consonants will be used as negative examples. In a similar manner, LEP-W will learn how to recognize a closed syllable.

To classify the **stress level** of each syllable in a word is difficult because in English stress is variable, i.e., it may occur on any syllable [Kreidler, 1989]. Nonetheless, rules can be identified concerning the placement of stress. Consider the words **record**, **indent**, **import**, and **export**, each of which has two different stress patterns. If they are nouns, the stress is on the first syllable, otherwise the stress is on the second syllable. By studying large samples of words in an on-line pronunciation dictionary, we anticipate that LEP-W will be able to identify rules governing the placement of stress.

The most important step in LEP-W is learning pronunciation rules for each English grapheme because the translation is performed grapheme by grapheme, syllable by syllable. Once LEP-W has learned all the pronunciation rules for each grapheme according to its conditions, such as **open** or **closed** syllable, translation is reduced to a simple matching problem. The component of LEP-W that will address this problem is LEP-G (Learning English Pronunciation for Graphemes), which is described in detail in this paper.

LEP-W will accumulate all the rules learned from the above steps and also have to learn to arrange them according to priority, delete redundant rules, and combine rules as necessary.

Having given a general description of how LEP-W learns to translate English words to IPA symbols, we now describe the specific problem, learning pronunciation rules for English graphemes.

We selected the following 12 graphemes for the learning experiment: **a**, **e**, **i**, **o**, **u**, **b**, **c**, **d**, **ar**, **au**, **or** and **gh**. This subset was chosen to include all the single-vowels, which are the most difficult graphemes for

Figure 1: Grapheme a and Three IPA Symbols

which to choose a pronunciation because each of the vowel graphemes represents more than one sound while most of the consonant graphemes only represent one sound. There are three consonant graphemes in the list, which were chosen alphabetically. We also chose four vowel graphemes, each consisting of two letters. For each grapheme, there is at least one corresponding IPA symbol. The main idea is to capture the relationship between a grapheme and its IPA symbol and record this relationship in rule form. A relationship is described as a set of conditions on the syllable containing the grapheme. In Figure 1, we show the relationship between the grapheme **a** and the IPA symbols [ei], [ash], and [schwa].

The LEP-G learning algorithm is based on the Version Space algorithm (VSA) [Mitchell, 1982, Winston, 1992] with our modifications. For each grapheme and its target IPA symbol, we input a set of positive and negative examples. LEP-G chooses, from the version space, the single hypothesis that is consistent with these examples, and this hypothesis becomes one of the pronunciation rules. Before a rule is saved to the database, LEP-W will check whether the rule is redundant or more general than other rules for the same grapheme, and either deletes the redundant rule or combines the more general rule with the others.

The remainder of this paper is organized as follows. We introduce our method with a detailed example

in Section 2. Then we present our adaptation of the Version Space algorithm for the problem of learning to pronounce English graphemes in Section 3. In Section 4, we describe the empirical results obtained by running the implemented version of our approach. In Section 5, we present our conclusions and suggest directions for future research. A detailed algorithm for our modified Version Space algorithm is given in Appendix A. Output and diagrams showing the version spaces for all examples are given in Appendix B. The source code for the Prolog program that implements our method is given in Appendix C. A table giving all possible graphemes in the English language is given in Appendix D

2 Descriptive Example

Suppose we want to learn when to pronounce the grapheme **a** with the sound denoted with the IPA symbol [ash] [Pullum and Ladusa, 1986] from a series of positive and negative examples. In this case, a *positive example* is a word which has the grapheme **a** and its IPA symbol is [ash]; a *negative example* is a word which has the grapheme **a** but its IPA symbol is not [ash]. The words **hat**, **lab**, **mad**, and **sad** are positive examples, and the words **make**, **station**, and **late** are negative examples. The first grapheme **a** in the word **capital** is a positive example, while the second **a** is a negative example. We restrict the set of negative examples to words that include the grapheme to be pronounced, i.e., **house** is not a negative example for learning to pronounce the grapheme **a** because **house** does not contain **a**.

The format of the examples is:

[IPA, GraphemeB, GraphemeA, TypeOfS, PartOfSpeech, Stress, NumOfSyllables]

Where

- IPA is the IPA symbol to be learned
- GraphemeB is the grapheme before the target grapheme
- GraphemeA is the grapheme after the target grapheme
- TypeOfS tells whether a syllable is open or closed
- Stress tells whether the syllable is stressed or unstressed

- NumOfSyllables is the number of syllables in the word

The word **hat** is a closed, one-syllable noun and the syllable containing the grapheme **a** (the only syllable) is stressed. Its input form is:

[ash, h, t, closed, noun, stressed, one-syllable]

No indication that it is a positive example is present in the input form. Whether or not an example is a positive example is determined by the learning procedure depending on the learning task.

The hypothesis space for this learning problem is the set of tuples containing seven fields: IPA symbol, the grapheme before and after the target grapheme, type of syllable, part of speech, stress, and the number of syllables. Each of these fields contains either a specific value or a question mark **?**. A specific value in a field denotes a required condition for that field, but a **?** means no restriction. We present the following two examples, where Example 1 uses the ordinary VSA and Example 2 uses our simplified VSA.

The target rule we want to learn is that the grapheme **a** in a closed, stressed syllable is pronounced as [ei]. In the output format, the target rule is represented as [ei, ?, ?, open, ?, stressed, ?]. The example words are: **cake**, **name**, **map**, **ate**, **banana** (the first grapheme **a**), **candidate** (the last grapheme **a**), and **cat**.

Let P_i denotes an positive example, and N_i denotes a negative example, where **i** is an index over the number of positive or negative examples. In the input format, these examples are:

```
P1: [ei, c, k, open, noun, stressed, one-syllable]

P2: [ei, n, m, open, noun, stressed, one-syllable]

N1: [ash, m, p, closed, noun, stressed, one-syllable]

P3: [ei, empty, t, open, verb, stressed, one-syllable]

N2: [schwa, b, n, open, noun, unstressed, multi-syllable]

P4: [ei, d, t, open, noun, stressed, multi-syllable]

N3: [ash, c, t, closed, noun, stressed, one-syllable]
```

Example 1:

For a straightforward application of the VSA, we begin with the most general hypothesis as [?, ?, ?, ?, ?, ?, ?] indicating the initial general hypothesis. The first positive example is used as the initial specific hypothesis.

Figure 2: Initial Version Space

We assume that the first example will always be a positive example. The initial specific hypothesis is thus the same as P1 [ei, c, k, open, noun, stressed, one-syllable]. The initial version space is shown in Figure 2, with the initial general hypothesis at the top and the initial specific hypothesis at the bottom. We continue looking at the examples one by one until all have been examined.

The second example is P_2 = [ei, n, m, open, noun, stressed, one-syllable], a positive example, which indicates that the current specific hypothesis (CSH) [ei, c, k, open, noun, stressed, one-syllable] should be generalized because it is too specific to include this positive example. The idea of generalization is to make minimal changes to the old specific hypothesis to create a more general hypothesis such that both P_1 and P_2 are instances. That is, we only change those fields in the tuple where P_1 and P_2 are different. The revised space is shown in Figure 3, with the new specific hypothesis labelled as NEW.

The next example is N_1 = [ash, m, p, closed, noun, stressed, one-syllable], a negative example. which requires us to make the general hypothesis more specific. Again, only minimal changes are made. We have to make sure that no new general hypothesis is a generalization of some other general hypothesis, no new general hypothesis is a generalization of the negative example N_1, and no new general hypothesis has a field which is more specific than the current specific hypothesis [Winston, 1992].

Now, let **CSH** denote the current specific hypothesis, **CGH** denote the current general hypothesis, and F_i denote each field in **CSH**, **CGH**, or N_j (negative examples), where **i** is an index of fields and **j** is an index of the negative examples. There are three conditions under which no new hypothesis is generated from field

Figure 3: Version Space after P_2

F_i:

- condition 1: F_i of N_j and F_i of **CSH** are the same
- condition 2: F_i of **CGH** and F_i of **CSH** are the same
- condition 3: F_i of N_j is equal to ?

Otherwise we generate a new hypothesis from this field by changing F_i of **CSH** to F_i of **CGH**, and leave all other fields unchanged.

Let us compare N_1 with each of **CSH** and **CGH**, field by field. The first field (F_1) of N_1 is **ash**, of **CSH** is **ei**, and of **CGH** is **?**. None of the three conditions applies here. Therefore, one new general hypothesis is generated by replacing **?** of **CGH** with **ei** and copying all other fields to form [ei, ?, ?, ?, ?, ?, ?]. The second field (F_2) of N_1 is **m**, but F_2 of **CSH** and **CGH** are both **?**. By condition 2, no new general hypothesis is produced from this field. The third field (F_3) of N_2, **CSH**, and **CGH** also satisfy condition 2, and therefore no new general hypothesis is generated from F_3 either. F_4 of N_1 is **closed**, of **CSH** is **open**, and of **CGH** is **?**. This situation does not match any of the three conditions, so a new hypothesis [?, ?, ?, open, ?, ?, ?] is generated by replacing **?** of **CGH** with **open** and copying all the rest of the fields. The last three fields of N_1 and **CSH** are the same, which satisfies condition 1, and therefore no new hypotheses are generated.

The resulting version space, shown in Figure 4 has two general hypotheses, which are labelled NEW1 and NEW2 in the figure.

Figure 4: Version Space After N_1

The next example is P_3 = [ei, empty, t, open, verb, stressed, one-syllable]. Only F_5 of P_3 is different from F_5 of **CSH**. In that field, **CSH** has **noun**, while P_3 has **verb**. Therefore, a new specific hypothesis is generated by replacing **noun** with **?** giving [ei, empty, t, open, ?, stressed, one-syllable] as shown in Figure 5.

Following P_3, we have a negative example N_2 = [schwa, b, n, open, noun, unstressed, multi-syllable]. For each of the two **CGH**, we do the same steps as we did for N_1. First consider the hypothesis CGH_1 [ei, ?, ?, ?, ?, ?, ?]. The F_1 of N_2 is **schwa**, of **CSH** is **ei**, and of **CGH** is **ei** . By condition 2, no new hypothesis is generated from F_1, because F_2 and F_3 of **CSH** and CGH_1 are the same. Next, F_4 of N_2 and **CSH** are the same. According to condition 1, no new hypothesis is generated from this field. Since F_5 of **CSH** and **CGH** are the same, by condition 2, no new hypothesis is generated from this field. The next field F_6 does not satisfy any of the three conditions. Therefore, a new hypothesis [ei, ?, ?, ?, ?, stressed, ?] is generated. Because none of the three conditions is true for F_7, a new hypothesis [ei, ?, ?, ?, ?, ?, one-syllable] is generated. Thus, from the hypothesis [ei, ?, ?, ?, ?, ?, ?] and the negative example [schwa, b, n, open, noun, unstressed, multi-syllable], we have generated two new hypotheses [ei, ?, ?, ?, ?, stressed, ?] and [ei, ?, ?, ?, ?, ?, one-syllable].

Figure 5: Version Space After P_3

So far, we have developed one of the two current general hypotheses. For the second hypothesis CGH_2 [?, ?, ?, open, ?, ?, ?], we repeat the same operations as for the first and generate hypotheses [ei, ?, ?, open, ?, ?, ?], [?, ?, ?, open, ?, stressed, ?], and [?, ?, ?, open, ?, ?, one-syllable].

The new version space after N_2 is shown in Figure 6.

Our last positive example is P_4 = [ei, d, t, open, noun, stressed, multi-syllable]. Since F_7 of P_4 and **CSH** have different value, we replace the value in F_7 of **CSH** with **?** giving a new **CSH**. So far, we have five **CGH**s (CGH_1, CGH_2, ..., CGH_5) and the new **CSH**. However, some of the **CGH**s fail to match the **CSH**. For example, CGH_2 = [ei, ?, ?, ?, ?, ?, one-syllable] and CGH_5 = [?, ?, ?, open, ?, ?, one-syllable] do not match **CSH** = [ei, ?, ?, open, ?, stressed, ?] because F_7 of each **CGH** has the value **one-syllable** while **CSH** has **?** in the same field. The value **?** matches anything and thus is more general than any specific value. Therefore, CGH_2 and CGH_5 are pruned away. A similar pruning operation is done after each example is learned. In previous examples, there were no such **CGH**s to be pruned away. The new version space after P_4 is shown in Figure 7.

The last negative example is N_3 = [ash, c, t, closed, noun, stressed, one-syllable], another negative

Figure 6: Version Space After N_2

Figure 7: Version Space After P_4

example. We have three **CGH**s from the previous version space. For CGH_1, only one new general hypothesis is produced from F_4 since **CSH** and **CGH** have the same value in the other fields. The new CGH_1 is [ei, ?, ?, open, ?, stressed, ?]. For CGH_2, the new **CGH** is a copy of the old one because F_{1-5} of **CSH** and **CGH** are the same, F_6 of **CSH** and N_3 are the same, and F_7 of **CSH** and **CGH** are the same. By condition 2, no new general hypothesis is produced from $F_{1-5,7}$ and by condition 1, no new general hypothesis is produced from F_6. So, the new CGH_2 is [ei, ?, ?, open, ?, ?, ?]. The new CGH_2 is a more general hypothesis than the new CGH_1 because F_6 of the new CGH_2 is **?** but F_6 of the new CGH_1 is **stressed**, and therefore the new CGH_2 is pruned away. From the last **CGH**, a new CGH_3 is generated but it is identical with the new CGH_1 and therefore, we prune it away. No pruning operations were needed for the previous negative examples because no new **CGH**s were more general than or identical with the other new **CGH**s. We present the final version space in Figure 8.

Our explanation so far uses the ordinary VSA. Note that the general models beginning with '?' are never used in the final solution because the first field matches any negative example. Therefore for our example, we can restrict the initial most general model to [ei, ?, ?, ?, ?, ?, ?].

Example 2:

For our modified VSA, we initialize the most general hypothesis as [ei, ?, ?, ?, ?, ?, ?]. The initial specific hypothesis is the first positive example P_1, as with the ordinary VSA. The initial version space, which is shown in Figure 9, is a subspace of the one shown in Figure 2. For Example 2, we perform the same operations for every positive and negative example that we did for Example 1. As a result, the final version space for Example 2, which is shown in Figure 10, is a subspace of the final version space for Example 1, which was shown in Figure 8.

Some of the advantages of the modified VSA algorithm are clear from the examples just given. With MVSA, the version space is smaller than with VSA: fewer nodes (9 instead of 11) are present and the graph is less complicated. As well, fewer levels are required in the version space (6 instead of 7) and the maximum width of the version space, which measures the maximum storage requirements, is reduced (2 instead of 3). Although the gains are small with this simple example, they become correspondingly greater with more complicated version spaces.

Figure 8: Final Version Space

Figure 9: Initial Version Space with MVSA

Figure 10: Final Version Space with MVSA

3 Method

The previous section illustrated how the VSA could be used to learn pronunciation rules for English graphemes. Mitchell declared "The version space method is assured to find all generalizations that are consistent with the observed training instances" [Mitchell, 1982], and the pronunciation rules found by LEP-G.1 are consistent with the training examples. If LEP-G.1 was trained with a bigger set of examples such as an one-line dictionary, the pronunciation rules found by LEP-G.1 would be consistent with all the words in that dictionary.

The following description of the Version Space algorithm is adapted from that given in [Winston, 1992]:

Initialization:

Step 1: Initialize the general hypothesis by the most general possible hypothesis and initialize the specific hypothesis by the first positive example.

Do the following steps until the set of examples is empty:

Step 2: If input is a positive example, then

(a) Generalize all specific models to match the positive example, but ensure the following:

(i) The new specific models involve minimal changes.

(ii) Each new specific model is a specialization of some general model.

(iii) No new specific model is a generalization of some other specific model.

(b) Prune away all general models that fail to match the positive example.

Step 3: If input is a negative example, then

(a) Specialize all general models to prevent matching the negative example, but ensure the following:

(i) The new general models involve minimal changes.

(ii) Each new general model is a generalization of some specific model.

(iii) No new general model is a specialization of some other general model.

(b) Prune away all specific models that match the negative example.

In general, the Version Space algorithm is easy to understand and reasonably efficient, but constraints in our problem allow some improvements in efficiency. Our modified Version Space algorithm (MVSA) saves at least half of the CPU time. In Section 2, we explained how the VSA could be applied to the problem of learning English pronunciation rules for graphemes. Example 1 uses the above VSA without modification and Example 2 uses MVSA, as described below and presented in detail in Appendix A.

There are three major modifications in MVSA. The first one is to avoid generating examples with an unconstrained value for the IPA symbol by choosing [IPA, ?, ?, ..., ?] as the most general hypothesis, instead of [?, ?, ?, ..., ?]. This modification greatly reduces the size of the version space; in the example in Section 2, 5 out of 6 second level hypotheses became unnecessary as a result of this modification.

The second modification is to restrict the number of specific hypotheses to one. For **LEP-G.1**, we input one example at a time. When the example is positive, specialization takes place and a new generation of specific hypotheses is generated. On one hand, the new specific hypothesis involves minimal changes from the old generation; on the other hand, it has to be a generalization of the input positive example. Suppose, the current specific hypothesis is [ei, m, d, open, verb, stressed, one-syllable], and the input positive example is [ei, c, k, open, noun, stressed, one-syllable]. We want to make minimal changes to produce the new general hypothesis. The new specific hypotheses would be the following:

[ei, ?, d, open, verb, stressed, one-syllable]

[ei, m, ?, open, verb, stressed, one-syllable]

[ei, m, d, open, ?, stressed, one-syllable]

which obviously are not generalization of the input positive example. The new specific hypothesis which is both involving minimal changes and being a generalization of the positive example has to be [ei, ?, ?, open, ?, stressed, one-syllable]. It is impossible to have more than one specific hypothesis at a time because of the nature of the specific hypothesis. For this reason, we leave out all operations involving more than one specific hypothesis, such as pruning away each new specific hypothesis which is a generalization of some other specific hypothesis. Thus, we delete the operation required by part (iii) in Step 2 of the VSA.

The third modification is to delete the pruning away of all specific hypotheses that match the negative example, i.e., part (b) of Step 3 of the VSA. Since by the first modification, the initial most general hypothesis

is of the form [IPA, ?, ?, ?, ?, ?, ?], the IPA value of all relevant specific hypotheses is fixed. Also, all negative examples must have an IPA value different from the IPA value in the positive examples because this is how we defined the negative examples. For this reason, we know the specific hypothesis will never match any negative examples because of the fixed IPA value. Therefore, we do not need to worry about pruning them away.

4 Results

The Version Space algorithm described in Section 3 is easy to understand, efficient, and easy to apply. We implemented it with the modifications described in Appendix A in a Prolog program called LEP-G.1. The implemented version learned to translate 12 English graphemes into IPA symbols and to generate 20 pronunciation rules.

Table 1 summarizes the results of running the LEP-G.1 program. In this table, the first column gives the grapheme, and it is followed by the run number (output number). The third column is the target IPA symbol which is learned. The fourth and fifth column show the grapheme before and after the target grapheme. The next column gives information about the openness of the syllable containing the target grapheme. The seventh column records the part of speech of the example word . The level of stress of the syllable containing the target grapheme is recorded in the eighth column. The last column gives the number of syllables in the target word. The first line of the table corresponds to the example discussed in Section 2, i.e., the rule that when the grapheme **a** occurs in an open, stressed syllable it is pronounced [ei]. Complete output and a diagram of the version space for each example is presented in Appendix B.

Table 2 gives statistics for our results. In this table, the first column gives the grapheme to be pronounced. The second column gives the IPA symbol. The next two columns record the number of negative and positive examples. The fifth column gives the maximum number of hypotheses. The sixth column records the number of *active examples*, i.e., those examples that result in changes to the version space. The last column tells what type of solution we obtained. Once again, the first line of the table corresponds to the example discussed in Section 2.

We now discuss the most interesting results obtained. In general, LEP-G.1 finds three different kinds of

Grapheme Learned	Output Number	IPA Symbol	Grapheme Before	Grapheme After	Open-ness	Part of Speech	Stress of Syllable	Number of Syllables
a	1	ei	?	?	open	?	stressed	?
	2	ash	?	?	closed	?	stressed	1
e	3	ϵ	?	?	closed	?	stressed	1
	4	ii	?	empty	open	?	stressed	1
i	5	i	?	?	closed	?	?	?
	6	ai	?	?	open	?	stressed	?
o	7	open_o	?	?	closed	?	stressed	1
	8	o	?	?	open	?	stressed	?
u	9	inv	?	?	closed	?	stressed	1
	10	u	?	?	open	?	stressed	1
b	11	b	?	?	?	?	stressed	?
	12	silent	m	empty	closed	?	?	?
c	13	k	empty	u	?	?	?	?
	14	k	?	l	?	?	stressed	?
	15	s	?	?	open	?	?	?
d	16	d	?	?	?	?	?	?
ar	17	ar	?	?	?	?	stressed	?
au	18	ash	l	gh	?	?	stressed	?
or	19	open_o_r	?	?	?	?	stressed	?
gh	20	silent	?	empty	closed	?	?	?

Table 1: Summary of Results

Grapheme Learned	IPA Symbol	Number of Negative Examples	Number of Positive Examples	Maximum Number of Hypotheses	Number of Active Examples	Type of Solution
a	ei	3	4	2	7	unique
	ash	2	4	2	5	multi-upper-bound
e	ϵ	2	4	2	4	single-upper-bound
	ii	5	3	6	6	multi-upper-bound
i	i	2	4	1	4	unique
	ai	4	3	1	4	unique
o	open_o	4	5	1	5	single-upper-bound
	o	4	4	2	5	single-upper-bound
u	inv	3	4	2	5	single-upper-bound
	u	3	3	1	3	single-upper-bound
b	b	3	5	1	2	single-upper-bound
	silent	5	7	3	5	multi-upper-bound
c	k	4	5	3	7	unique
	k	6	4	2	4	unique
	s	4	6	2	5	unique
d	d	0	6	1	3	unique
ar	ar	2	5	1	3	unique
au	ash	5	3	1	4	single-upper-bound
or	open_o_r	4	7	1	4	unique
gh	silent	3	5	3	6	unique

Table 2: Statistics for Results

solutions depending on how many possible hypotheses remain after all examples have been processed. First, consider the solution shown in Figure 10. We can see that the final general hypothesis and the final specific hypothesis are identical. We call this type of solution *unique solution*. As indicated by the last column of Table 2, unique solutions were found for many graphemes; details can be found in the Appendix B.

Now, let us compare the last generations of Output2 in Figure 11 and Output3 in Figure 12. Output2 has two general hypotheses while Output3 has only one. They have one thing in common, i.e., every final general hypothesis is a generalization of the specific hypothesis. Should we use the general hypothesis or the specific hypothesis as our final solution? The general hypothesis may seem preferable because it includes more cases than the specific hypothesis. Theoretically, this is correct, but it is not suitable for our application. Since the examples selected for learning a particular grapheme comprise only a small part of the applicable examples in a dictionary, some counterexamples may exist elsewhere in the dictionary. Therefore, we call this type of solution an *open solution set*, i.e., a set of solutions including possibly many general hypotheses as the upper bound and a specific hypothesis as the lower bound. If only one general hypothesis is present, the solution is a *single upper bound solution*, and otherwise it is a *multi upper bound solution*. The solution set for Output3, as shown in Figure 12, is an example of a single upper bound solution, and the solution set for Output2 (Figure 11) is a multi upper bound solution. As more examples are examined, an open solution set may be constrained to a unique solution. Similarly, a multi upper bound solution may be constrained to a single upper bound solution or possibly a unique solution.

Now let us examine Output11 and Output12 (as given in Appendix B). The purpose of these learning runs is to produce a pronunciation rule for the grapheme **b**. There are two cases for the pronunciation of the grapheme **b**.

Case 1: [b] in **basic**, **cube**, **rub**, **blue**, and **blackboard**

Case 2: [silent] in **aplomb**, **bomb**, **climb**, **comb**, **thumb**, and **coxcomb**

For Output11, we use Case 1 as positive examples and Case 2 as the negative examples. The result is an overly general solution [b, ?, ?, ?, ?, stressed, ?] because as a pronunciation rule it will match the words **climb**, **comb**, and **bomb** which should be pronounced as [silent] instead of [b]. That is, although [b, ?, ?, ?, ?, stressed, ?] as a solution does not match the negative examples since all negative examples have the value

Figure 11: Final Version Space for Output2

Figure 12: Final Version Space for Output3

[silent] in the first field, it is not a correct pronunciation rule for the grapheme **b** when **b** follows grapheme **m**.

We get such a solution because we did not pay attention to which case is more general and which case is more specific. For this problem, the more general case has more examples than the more specific case; for example, in the dictionary of Unix's `spell` program there are only 23 English words ending with **mb** in which **b** is [silent], and 1,237 English words starting with **b** which is pronounced as [b]. First we should learn the more specific case (the silent **b**) and then we should learn the less specific case (the [b] sound). Using this variation, we produced Output12, in which the solution is [silent,m,empty,closed,?,?,?], which means that if **b** is following **m** and nothing is following it, then **b** is silent. This is a good rule for pronouncing **b**. For the rest of the English words with the grapheme **b**, we use the general rule from Output11. LEP-G.1 will accumulate all the rules as the learning process progresses and rearrange them according to their priorities. The more specific a rule is, the higher its priority is. That is, the pronunciation rule for **b** in Output12 has higher priority than that in Output11.

With regard to choosing specific and general cases for a grapheme with two IPA symbols, we use the IPA symbol that has the fewest words as the specific case and the other IPA symbol as the general case. When pronunciation rules are used in an English-to-IPA translator, the one which has more fields with specific values has higher priority. We have not yet investigated the ordering of cases for graphemes with many IPA symbols.

Output13, Output14, and Output15 (see Appendix B) gave the most interesting results. The solution for Output13 is [k, empty, u, ?, ?, stressed, ?], which means that if the grapheme **c** is at the beginning of a word, it is followed by the grapheme **u**, and it is in a stressed syllable, then, **c** is pronounced with the [k] sound. For Output14, the solution is [k, ?, l, ?, ?, stressed, ?], which means that if the grapheme **c** is followed by **l** and it is in a stressed syllable, then it is pronounced with the [k] sound. In Output15, a remarkably simple rule was found for when **c** is to be pronounced with an [s] sound. The example words are: **cancer**, **cat**, **race**, **edict**, **edifice**, **camp**, **dance**, **candidate**, and **cyclist**. The solution is [s,?,?,open,?,?,?], which means that **c** is pronounced as an [s] sound whenever the syllable is open. The rules found in each of Output 13 and 14 are much simpler than the rules created by hand for the English-to-IPA translator described in [Zhang, 1993b].

Instead of 14 rules for the grapheme **c**, only three rules are needed, assuming that more complex conditions are allowed. In this case, we require that the field **After** be able to distinguish types of vowels, such as low or back vowels or blend consonants, instead of simply identifying the next grapheme. Although LEP-G.1 does not have the ability to extend its learning fields for the hypotheses, the quality of the rules it found for the grapheme **c** clearly shows that LEP-G.1 uses a suitable method for forming pronunciation rules.

5 Conclusions and Research Directions

So far LEP-G.1 has learned 12 graphemes and produced 20 pronunciation rules from 20 groups of English words. There are total of 64 different graphemes in English which we have summarized in Appendix D. The experiment where LEP-G.1 learned pronunciation rules for 12 graphemes strongly suggests that learning the other 52 graphemes is feasible. As well, learning pronunciation rules for English seems possible, because all English words can be decomposed into individual graphemes. Further experimentation and research is needed in order to reach this goal.

The learning program LEP-G.1 does not store a pronouncing dictionary in the database, instead it accumulates the pronunciation rules that it has learned from a group of English words. Therefore, it is more efficient in terms of space and allows pronunciation of unseen words.

Further experimentation is required to check the pronunciation rules found. In particular, the rules for the grapheme **c** should be checked against a complete dictionary.

LEP-G.1 is only part of the solution to the task of learning pronunciation rules for graphemes. It needs for automatic classification of examples as negative or positive and exception handling, as described below.

THE LEP-G.1 program should be augmented with the ability to classify examples. Recall that a positive example has the target grapheme and its sound is represented by certain IPA symbol, while a negative example also has the same target grapheme but its sound is represented by a different IPA symbol. Suppose that LEP-G.1 reads in 100 words and wants to learn to pronounce the grapheme **a**, and there are 10 of the 100 words with **a** pronounced as [ei], 30 of them with **a** pronounced as [ash], and rest of them with **a** pronounced as [schwa]. This means there will be 3 rules produced and one for each group. LEP-G.1 should take the first group as positive examples and the other two groups as negative examples for the first

rule. Then it should take the second group as positive examples and the rest of the two groups as negative examples for the second rule. Lastly, it should take the third group of words as the positive examples and the rest as the negative examples. With this augmentation, LEP-G.1 would be able to classify the negative and positive examples automatically.

Exception handling is required to deal with a minority pronunciation in a group of words, where no rule can be formed using the available conditions. For each word with a minority pronunciation, an *exception rule* should be generated for it. Where possible exception rules should be combined. For example, the grapheme **ear** is usually pronounced as [rhs] (right-hook schwa) [Pullum and Ladusa, 1986], such as **dear**, **ear**, **fear**, **beard**, **year**, and **hear**; but it pronounced as [ϵ_r] in the words **bear**, **pear**, and **wear**. In the UNIX `spell` dictionary, there are 3,318 one-syllable words; 28 of these words include the grapheme **ear**, and among these 28 words there are only the three exceptions mentioned above. LEP-G.1 will have to be augmented to create three rules for these exception words. An alternate approach is to augment the conditions available for use in the rules. For example, if information were added about the articulation point of consonants, then the three consonants **f**, **p**, and **w** would be classified as **bilabial** consonants and the other consonants would be classified as nonbilabial[O'grady and Dobrovolsky, 1992]. With this information, a single pronunciation rule for the three exceptions could be created: **ear** is pronounced [ϵ_r] whenever it follows a bilabial consonant and ends the syllable.

As we stated in Section 1, LEP-G is one of seven components of the learning procedure LPE-W. When complete, LPE-W will consists of the following components: recognizing grapheme(s) of a word, separating the word between syllables, distinguishing **open** and **closed** syllables, classifying the **stresses** of each syllable in a word, learning pronunciation rules for each grapheme, accumulating the pronunciation rules that has been learned, and a database.

Let us briefly describe one of these components, learning to classify stresses, using the MVSA. The learning procedure is called LCS-S (learning to classify stresses for syllables). There will be two necessary conditions for the LCS-S to learn classification of stresses: one is to use the part of speech and the other is to use information about the number of syllables. Some words such as **record** have the stress on a different syllable when it is a different part of speech. As a noun, it is stressed on the first syllable, but as a verb, it is

stressed on the second syllable. Therefore, the information on part of speech is essential. Also, the number of syllables is another necessary condition since stresses are on different syllables depending on the number of syllables. Therefore, the LCS-S will have the following form for each hypothesis in the version space: [P, Ns, Ps, Ss, Un] where **P** stands for the **part of speech of the word**, **Ns** stands for the **number of syllables**, and **Ps**, **Ss**, or **Ns** stands for **primary**, **secondary**, or **unstressed** syllable. Each of the fields which stands for stress will have a list of the position of syllables which the stress is on. For example, the word **antibiosis** is syllabicated as **an-ti-bi-o-sis** and the stresses of this word can be represented by [noun, 5, [1], [4], [2,3,5]]. Given pairs of words and their syllabicated forms, LCS-S will generate syllabication rules.

Our work with LEP-G has demonstrated that an algorithm based on the Version Space algorithm can learn pronunciation rules for English graphemes. A similar approach appears promising for the general problem of learning pronunciation for phonetic languages.

References

[Kreidler, 1989] Kreidler, C. W. (1989). *Pronunciation of English.* Basil Blackwell, Reading, MA.

[Mackay, 1987] Mackay, I. R. A. (1987). *Phonetics: The Science of Speech Production.* Pro.ed, Reading, MA.

[Mitchell, 1982] Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, 18:203–226.

[Morris, 1991] Morris, I., editor (1991). *The American Heritage Dictionary.* Houghton Mifflin Company, Reading, MA.

[O'grady and Dobrovolsky, 1992] O'grady, W. and Dobrovolsky, M. (1992). *Contemporary Linguistic Analysis.* Copp Clark Pitman Ltd., Reading, MA.

[Pullum and Ladusa, 1986] Pullum, G. K. and Ladusa, W. A. (1986). *Phonetic Symbol Guide.* The University of Chicago Press, Reading, MA.

[Winston, 1992] Winston, P. (1992). *Artificial Intelligence, Third Edition.* Addison-Wesley, Reading, MA.

[Zhang, 1993a] Zhang, J. (1993a). Automatic learning of English pronunciation for words. Unpublished manuscript.

[Zhang, 1993b] Zhang, J. (1993b). An English to International Phonetic Alphabet translator: Final report. Unpublished manuscript.

Appendix A: Detailed MVSA

```
Step 1: Initialization

        General_hypothesis = [IPA, ?, ?, ?, ?, ?, ?]
        Specific_hypothesis = the first positive example

Step 2: Loop until all the examples are exhausted or reaching to a
unique solution

        (1). if input is a positive example, then

             (a). generalize the specific hypothesis
                  for each General_hypothesis

                  if    any field of the positive example is
                        different from the corresponding
                        field of Specific hypothesis
                  then  change the value of the field into '?'
                  else  leave the value of the specific hypothesis
                        unchanged

             (b). prune away those general hypothesis that are more
                  specific than the current specific hypothesis

                  if    any field of the specific hypothesis is '?',
                        and the corresponding field of the general
                        hypothesis has some more specific value
                  then  delete this general hypothesis from the
                        version space

             (c). prune away all general models that fail to match
                  the positive example.

                  if    any field of a general hypothesis has a
                        value which is different from the value in the
                        the same filed of the positive example and this
                        different value is not '?'
                  then  delete this general hypothesis

        (2). if input is a negative example, then

             (a). specialize all general hypothesis to prevent
                  matching the negative example

                  if    any field of the general hypothesis and the
                        corresponding field of the specific hypothesis
                        are the same
                or if   any field of the negative example and the
                        corresponding field of the specific hypothesis
                        are the same
                or if   any field of the specific hypothesis is '?'
```

```
          then  no new general hypothesis is produced
          else  copy this field from the specific hypothesis
                to the corresponding field of the new general
                hypothesis and copy the other fields from the
                old general hypothesis to the corresponding
                fields of the new general hypothesis

     (b). prune away those general hypothesis that are
          specialization of some other general hypothesis

          if    every field of one general hypothesis is either
                a '?' or the value is equal to the value in the
                corresponding field of another general hypothesis
          then  delete the second one which is more specific than
                the first one
          if    every field of one general hypothesis is either
                a '?' or the value is equal to the value in the
                corresponding field of another general hypothesis
(3). if   input is exhausted or we reach to a unique solution
     then output the solution and exit
     else do (1) to (3)
```

Appendix B: Output and Diagrams

Appendix C: Source Code

Appendix D: English Graphemes and Their IPA Symbols

Grapheme	IPA Representation	Examples		Grapheme	IPA Representation	Examples
b	b	boy bed		ch	ch	desk deep
ck	k	pick ticket		cc	ks	success succeed
c	g	scatter scared		c	k	class cry
c	s	city cycle		dg	zh	bridge badge
dr	dr	drill dream		ds	dz	hands kinds
d	d	bridge badge		f	f	five photo
ght	t	eight daughter		gh	f	photo tough
g	zh	cage baggage		g	g	glad egg
h	h	home hook		j	zh	jeep just
kn	n	knight knife		k	g	skip scatter
k	k	key pick kite		l	dark_l	ball hall
l	l	like light		m	m	map mean mine
ng	eng	bring king		n	n	nine knight
ph	ph	photo philosophy		p	b	spit spar
p	p	people pipe		qu	kw	quack quick
r	r	run rat read		s	yogh	television confusion
s	z	is shoes		s	s	sun sea
tch	ch	ditch catch		th	theta	think thick
th	eth	this that		tr	tr	tree train
ts	ts	rats roots		t	d	student steam
t	t	tent table		v	v	vowel five
w	w	woman work		x	z	xylene xylol
x	ks	box oxen		y	j	yard yellow
z	z	zoo zero				
ai	ei	plain aim		al	l_open_o	ball hall
ar	ar	car bar star		a	ei	cable age
a	schwa	machine banana		a	ash	map lamp
ear	ir	ear hear		eer	ir	beer deer
ere	ir	here mere		ea	ii	sea tea
ee	ii	see bee		ew	ju	new review
e	ii	be me		e	e	desk text
i	i	big tip		i	ai	bike five
oor	open_o_r	door poor		ow	au	how now
ou	au	house doubt		au	open_o	caught daughter
oi	open_o_i	oil soil		oo	uu	room stool
oo	u	book look		oy	open_o_i	boy soy
oa	ou	boat oat		ow	ou	bow bellow
o	ou	home note		o	open_o	dog not
ure	ur	endure sure		ur	schwa_r	fur nurse
ir	schwa_r	bird skirt		u	ju	cute huge
u	inv_v	cup truck				