INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Automatic Induction of Finite State Transducers for Simple Phonological Rules

Dan Gildea and Dan Jurafsky
International Computer Science Institute and
University of California at Berkeley
{gildea,jurafsky}@icsi.berkeley.edu

TR-94-052

October 1994

Abstract

This paper presents a method for learning phonological rules from sample pairs of underlying and surface forms, without negative evidence. The learned rules are represented as finite state transducers that accept underlying forms as input and generate surface forms as output. The algorithm for learning them is an extension of the OSTIA algorithm for learning general subsequential finite state transducers. Although OSTIA is capable of learning arbitrary s.f.s.t's in the limit, large dictionaries of actual English pronunciations did not give enough samples to correctly induce phonological rules. We then augmented OSTIA with two kinds of knowledge specific to natural language phonology, representing a naturalness bias from "universal grammar". A bias that underlying phones are often realized as phonetically similar or identical surface phones was implemented by using alignment information between the underlying and surface strings. A bias that phonological rules apply across natural phonological classes was implemented by learning decision trees based on phonetic features on each state of the transducer. The additions helped in learning more compact, accurate, and general transducers than the unmodified OSTIA algorithm. An implementation of the algorithm successfully learns a number of English postlexical rules, including flapping, t-insertion and t-deletion.

1 Introduction

Johnson (1972) first observed that traditional phonological rewrite rules can be expressed as regular relations if one accepts the constraint that no rule may reapply directly to its own output. Aside from the interesting result that the traditional formalism for expressing phonological rules is less powerful than it appears, this finding meant that finite state transducers can be used to represent phonological rules. Since parsing with finite state transducers is much simpler than parsing context-sensitive rewrite rules, this greatly simplified the problem of parsing the output of phonological rules in order to obtain the underlying, lexical forms (Karttunen 1993).

In this paper we explore another consequence of FST models of phonological rules: their weaker generative capacity also makes them easier to learn. We describe our preliminary algorithm and results on one method of learning rules from sample pairs of input and output strings.

2 Subsequential Transducers and Phonological Rules

Since Johnson's (1972) work, researchers have proposed a number of different ways to represent phonological rules by transducers. In order to take advantage of recent work in transducer induction, we have chosen to represent rules as *subsequential finite state transducers*. Subsequential finite state transducers are a subtype of finite state transducers with the following properties:

1. The transducer is deterministic, that is, there is only one arc leaving a given state for each input symbol.
2. Each time a transition is made, exactly one symbol of the input string is consumed.
3. A unique end of string symbol is introduced. At the end of each input string, the transducer makes an additional transition on the end of string symbol.
4. All states are accepting.

The length of the output strings associated with a subsequential transducer's transitions is not constrained. The subsequential transducer for the English flapping rule in 1 is shown in Figure 1; an underlying *t* is realized as a flap after a stressed vowel and any number of *r*'s, and before an unstressed vowel.

(1) $\mathrm{t} \rightarrow \mathrm{dx} \,/\, \acute{V}\ \mathrm{r}^{*}\ \underline{\quad}\ \mathrm{V}$

The most popular formalism for represent phonological rules as transducers is the *two-level* formalism of Koskenniemi (1983), based on Johnson (1972) and the (only recently published) work of Kaplan & Kay (1994), and the various implementations and extensions of the two level formalisms (summarized and contrasted in Karttunen (1993); we will henceforth refer to Karttunen's paper for details on two-level phonology). The most significant difference between this and our subsequential transducers is that the two-level transducers described by Karttunen (1993) are non-deterministic. In addition, Karttunen's transducers may have only zero or one symbol as either the input or output of an arc, and they have no special end of string symbol. Finally, Karttunen's transducers explicitly include both accepting and non-accepting states; our subsequential transducers represent the fail states only implicitly.

These representational differences between the two formalisms lead to different ways of handling certain classes of phonological rules, particularly those that depend on the context to the right of the affected symbol.

The subsequential transducer does not emit any output until enough of the right hand context has been seen to determine how the input symbol is to be realized. Notice in the transducer in Figure 1 that no output is emitted upon seeing a 't' when the machine is at state 1. Rather, the machine goes to state 2 and waits to see if the next input symbol is the requisite unstressed vowel; depending on this next input symbol, the machine will emit the 't' or a 'dx' along with the next input symbol when it makes the transition from state 2 to state 0.

In contrast, the non-deterministic two-level-style transducer shown in Figure 2 has two possible arcs leaving state 1 upon seeing a 't', one with 't' as output and one with 'dx'. If the machine takes the wrong

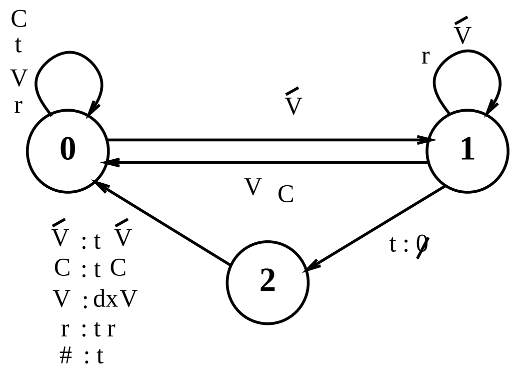


Figure 1: *Subsequential Transducer for English Flapping*: Labels on arcs are of the form (input symbol):(output symbol). Labels with no colon indicate the same input and output symbols. 'V' indicates any unstressed vowel, 'V́' any stressed vowel, 'dx' a flap, and 'C' any consonant other than 't', 'r' or 'dx'. '#' is the end of string symbol.

transition, the subsequent transitions will leave the transducer in a non-accepting state, or a state will be reached with no transition on the current input symbol. Either way, the transduction will fail.

Generating a surface form from an underlying form is more efficient with a subsequential transducer than with a nondeterministic transducer, as no search is necessary in a deterministic machine. Running the transducer the other way to parse a surface into possible underlying forms, however, remains non-deterministic in subsequential transducers. In addition, a subsequential transducer may require many more states than a non-deterministic transducer to represent the same rule (this will be discussed in further detail in §7.) Our reason for chosing subsequential transducers, then, is solely that efficient techniques exist for learning them, as we will see in the next section.

3 The OSTIA Algorithm

Our phonological-rule induction algorithm is based on augmenting the Onward Subsequential Transducer Inference Algorithm (OSTIA) of Oncina *et al.* (1993). This section outlines the OSTIA algorithm in order to provide background for the modifications described in the remainder of the paper. For further detail, see Oncina *et al.* (1993).

OSTIA takes as input a training set of input-output pairs. The algorithm begins by constructing a tree transducer which covers all the training samples according to the following procedure: A branch is added to the tree for each input string by following a path from the initial state of the machine with one arc corresponding to each symbol in the input string. When there is no move on the next input symbol from the present state, a new branch is grown on the tree. The entire output string of each transduction is initially stored as the output on the last arc of the transduction, that is, the arc corresponding to the end of string symbol. An example of an initial tree transducer constructed by this process is shown in Figure 3.

As the next step, the output symbols are "pushed forward" as far as possible towards the root of the tree. This process begins at the leaves of the tree and works its way to the root. At each step, the longest common prefix of the outputs on all the arcs leaving one state is removed from the output strings of all the arcs leaving

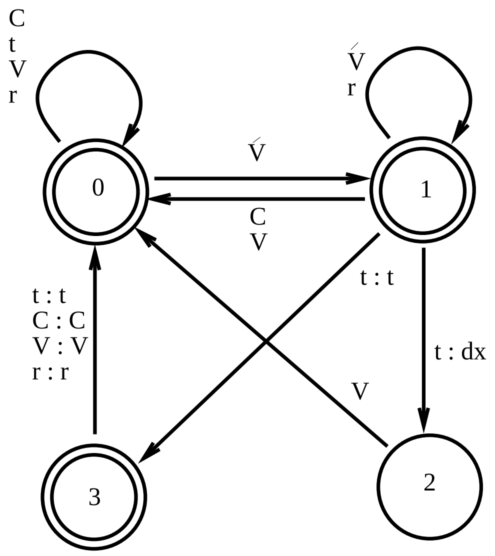


Figure 2: *Nondeterministic Transducer for English Flapping*

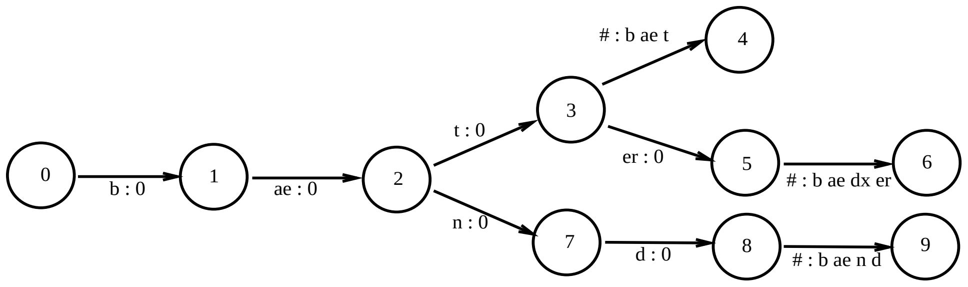


Figure 3: *Initial Tree Transducer for "bat", "batter", and "band" with Flapping Applied* ('dx' indicates a flap)

the state and suffixed to the (single) arc entering the state. This process continues until the longest common prefix of the outputs of all arcs leaving each state is the null string – the definition of an *onward* transducer. The result of making the transducer of Figure 3 onward is shown in Figure 4.

At this point, the transducer covers all and only the strings of the training set. OSTIA now attempts to generalize the transducer, by merging some of its states together. For each pair of states (s, t) in the transducer, the algorithm will attempt to merge s with t, building a new state with all of the incoming and outgoing transitions of s and t. The result of the first merging operation on the transducer of Figure 4 is shown in Figure 5.

A conflict arises whenever two states are merged that have outgoing arcs with the same input symbol. When this occurs, an attempt is made to merge the destination states of the two conflicting arcs. First, all output symbols beyond the longest common prefix of the outputs of the two arcs are "pushed back" to arcs further down the tree. This operation is only allowed under certain conditions which guarantee that the transductions accepted by the machine are preserved. The push back operation allows the two arcs to be combined into one and their destination states to be merged. An example of a push back operation and subsequent merger from a slightly extended domain than the earlier examples is shown in Figure 6. This method of resolving conflicts repeats until no conflicts remain, or until resolution is impossible. In the latter

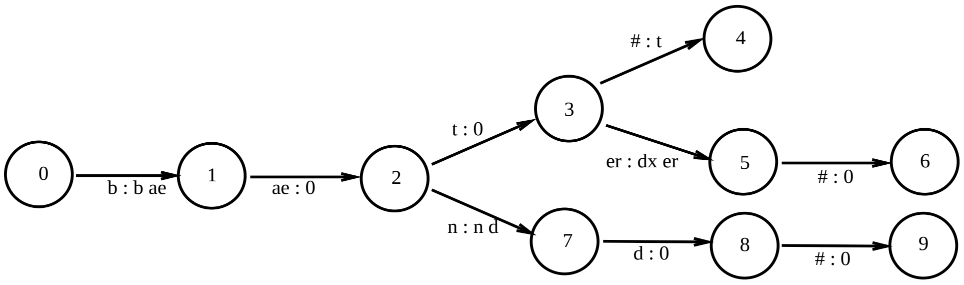


Figure 4: *Onward Tree Transducer for "bat", "batter", and "band" with Flapping Applied*

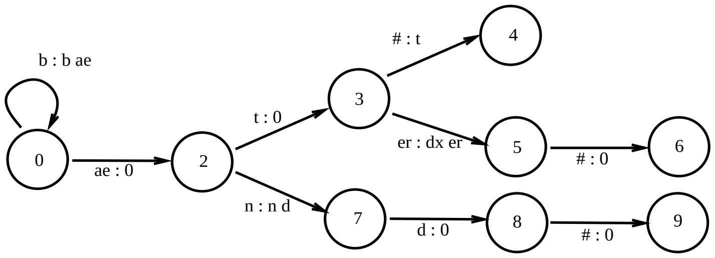


Figure 5: *Result of Merging States 0 and 1 of Figure 4*

case, the transducer is restored to its configuration before the merger causing the original conflict, and the algorithm proceeds by attempting to merge the next pair of states.

4 Problems Using OSTIA to learn Phonological Rules

The OSTIA algorithm can be proven to learn any subsequential relation in the limit. That is, given an infinite sequence of valid input/output pairs, it will at some point derive the target transducer from the samples seen so far. However, when trying to learn phonological rules from linguistic data, this result may be too weak. The necessary number of sample transductions may be several times the size of any natural language's vocabulary, or the necessary sample may require strings that are not found in the language. In particular, systematic phonological constraints such as syllable structure may make it impossible to obtain the set of examples that would be necessary for OSTIA to learn the target rule. For example, given a training set of examples of English flapping, the algorithm may induce a transducer that realizes an underlying 't' as 'dx' either in the environment $\acute{V} r^* \_\_ V$ or after any sequence of six consonants. This is possible since such a transducer will accurately cover the training set, as no English words contain six consonants followed by a 't'. The OSTIA algorithm is a very general one, making no assumptions about the input or output alphabets of the transducer,

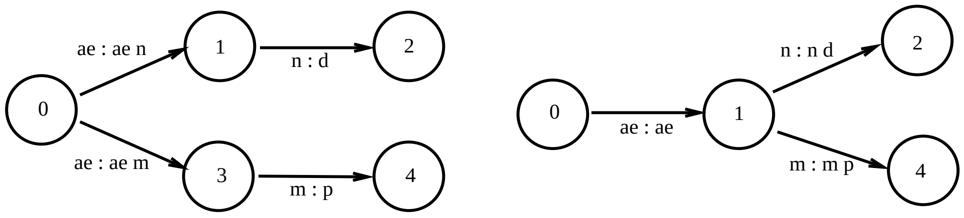


Figure 6: *Example Push Back Operation and State Merger*

or about the target relation, beyond the requirement of subsequentiality. Thus the algorithm does not have the language bias which would allow it to avoid such an unnatural transducer.

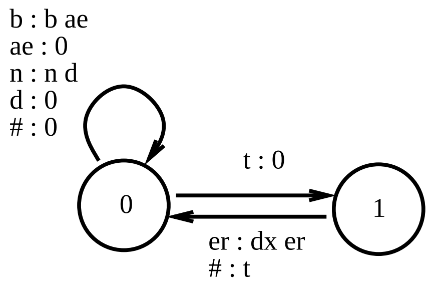


Figure 7: *Final Result of Merging Process on Transducer from Figure 4*

Another unnatural aspect to the transducers produced by OSTIA is their tendency toward "clumping". This is illustrated by the arcs with out "b ae" and "n d" in the transducer in Figure 7, or even Figure 4. Because OSTIA moves all the output symbols as far as possible towards the root of the initial tree, its default behavior is to emit the remainder of the output string for a transduction as soon as enough input symbols have been seen to uniquely identify the input string in the training set. While the push back operations performed during the process of merging states will cause most of the clumps of output symbols to be spread out, many will remain unless the training set covers all possible sequences of input symbols relatively densely. Using data from the lexicon of a natural language, such dense coverage is not likely. The clumping of output symbols results in machines which may, seemingly at random, insert or delete sequences of four or five phonemes, something which is linguistically implausible. In addition, the incorrect distribution of output symbols prevents the optimal merging of states during the learning process, resulting in large and inaccurate transducers.

OSTIA's lack of language bias also leads to cases in which it is unable to generalize from the input data. or generalizes in an unnatural manner, causing it to perform poorly on the test set. An example of an unnatural generalization is shown in 7, the final transducer induced by OSTIA on the three word training set of Figure 3. While this transducer does correctly handle the training examples, it is linguistically unnatural, and we expect it to do poorly on any test set. For example, the transducer of Figure 7 will insert an 'ae' after any 'b', and delete any 'ae' from the input. Perhaps worse, it will fail completely upon seeing any symbol other than 'er' or the end of string symbol after a 't'. While it might be unreasonable to expect any transducer trained on three samples to be perfect, the transducer of Figure 7 illustrates on a small scale a number of ways in which the generalizations of OSTIA algorithm may be made more natural linguistically.

In a number of other cases, OSTIA is unable to generalize at all. For example, a large number of the errors encountered when running transducers learned by OSTIA on test data are simply cases of falling off the transducer, that is, having no next arc for the next input symbol. For example the transducer of Figure 7 will fail on any word that does not begin with 'b', 'ae', 'n', or 't'. Again, the induced transducers must have a way to guess the next move in such cases by generalizing from observed data.

Similarly, if the OSTIA algorithm is training on cases of flapping in which the preceding environment is every stressed vowel but one, the algorithm has no way of knowing that it can safely generalize the environment to all stressed vowels. The algorithm needs knowledge about classes of phonemes (i.e., the concepts 'vowel' or 'stressed vowel') to fill in accidental gaps in training data coverage.

The next two sections summarize our work in adding two kinds of language bias to OSTIA. *Alignment* information is used to avoid the unnatural clumping phenomena, and *phonological feature* information is used to generalize rules in a phonological natural way.

5 Using Alignment Information

Our first modification of OSTIA was to add the bias that, all things being equal, the surface string of phones should resemble the underlying string of phones. That is, as a default, a phoneme is realized as itself, or as a phonologically similar phone.

This bias is a natural one for phonological strings; besides two-level phonology, this bias was also a fundamental tenet of natural phonology. As §4 discussed, OSTIA lacks such a bias by default since the technique of pushing the output symbols of the initial tree transducer as far forward as possible makes no assumptions about the correspondence between symbols in the input and output strings, although the merging process will often push the output symbols back to their correct places in the transducer.

In order to add this information, the algorithm guesses the most probable phoneme to phoneme alignment between the input and output strings, and uses this information to more sensibly distribute the output symbols among the arcs of the initial tree transducer.[1]

The modification proceeds in two stages. First, a dynamic programming method is used to distribute output symbols among the arcs of the tree transducer built at the start of the learning process. This is demonstrated for the word "importance" in Figures 8 and 9.

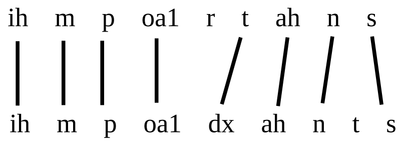


Figure 8: Alignment of "importance" with flapping, r-deletion and t-insertion

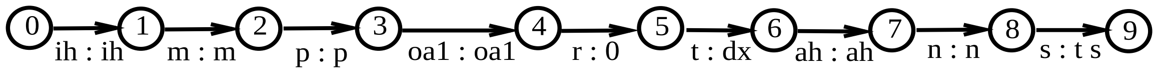


Figure 9: Resulting initial transducer for "importance"

The alignment uses the algorithm of Wagner & Fischer (1974), which calculates the insertions, deletions, and substitutions which make up the minimum edit distance between the underlying and surface strings. The costs of edit operations are based on phonetic features; we used 26 binary articulatory features. The cost function for substitutions was equal to the number of features changed between the two phonemes. The cost of insertions and deletions was 6 (roughly one quarter the maximum possible substitution cost). From the sequence of edit operations, a mapping of output phonemes to input phonemes is generated according to the following rules:

- Any phoneme maps to an input phoneme for which it substitutes
- Inserted phonemes map to the input phoneme immediately following the first substitution to the left of the inserted phoneme

Second, when adding a new arc to the tree, all the unused output phonemes up to and including those which map to the arc's input phoneme become the new arc's output, and are now marked as having been used. When walking down branches of the tree to add a new input/output sample, the longest common prefix, *n*, of the sample's unused output and the output of each arc is calculated. The next *n* symbols of the transduction's output are now marked as having been used. If the length, *l*, of the arc's output string is greater than *n*, it is

[1]By using the new distribution of output symbols along the arcs of the initial tree transducer, we are no longer guaranteed that it is onward. The onwardness of the transducer is an invariant of the unmodified algorithm, as indicated by the name OSTIA (Onward Subsequential Transducer Inference Algorithm). However, onwardness is not essential to the correctness of the algorithm, and even the final transducers induced by our new method tend to be onward.

necessary to push back the last *l* – *n* symbols onto arcs further down the tree. A tree transducer constructed by this process is shown in Figure 10, for comparison with the unaligned version in Figure 4.

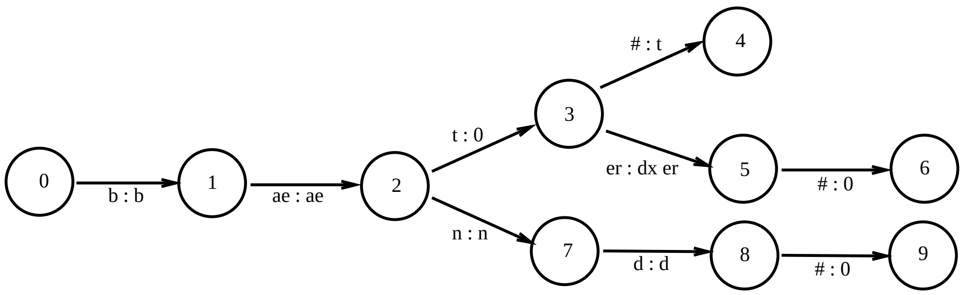


Figure 10: *Initial Tree Transducer Constructed with Alignment Information*: Note that output symbols have been pushed back across state 3 during the construction

Results of our alignment algorithm are summarized in §7. The distribution of output symbols resulting from the alignment constrains the merging of states early in the merging loop of the algorithm by making it less likely that the output symbols of conflicting arcs can be pushed back. Interestingly, preventing the wrong states from merging early on allows more merging later, and results in more compact transducers.

6 Generalizing Behavior With Decision Trees

In order to allow OSTIA to make natural generalizations in its rules, we added a decision tree to each state of the machine, describing the behavior of that state. For example, the decision tree for state 2 of the machine in Figure 1 is shown in Figure 11. Note that if the underlying phone is an unstressed vowel ([-cons,-stress]), the machine outputs a flap, followed by the vowel, otherwise it outputs a ‘t’ followed by the underlying phone.

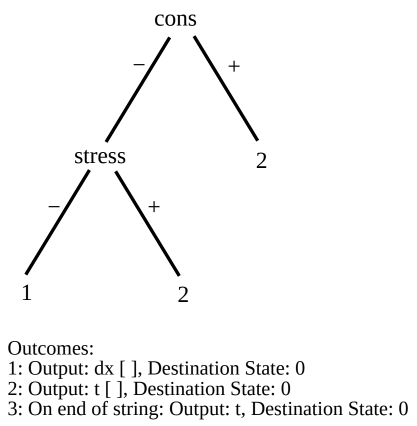

cons
–
+
stress
2
–
+
1
2
Outcomes:
1: Output: dx [ ], Destination State: 0
2: Output: t [ ], Destination State: 0
3: On end of string: Output: t, Destination State: 0

Figure 11: *Example Decision Tree*: This tree describes the behavior of State 2 of the transducer in Figure 1. [ ] in the output string indicates the arc’s input symbol (with no features changed).

The decision trees describe the behavior of the machine at a given state in terms of the next input symbol by generalizing from the arcs leaving the state. In order to to this, we need a way of describing the output of an arc in terms of its input symbol. This is quite straightforward if, for example, the output consists of one phoneme, identical to the input phoneme. Often, however, because of insertions or deletions or mere context dependencies, the output phoneme on an arc corresponds not to the arc’s input phoneme but rather to some position earlier in the string. The arc may also have null output, or may have an output string of several phonemes. In such cases, we wish to make generalizations such as “at this state, on any consonant input phoneme, emit ’a’ followed by the input consonant, and jump to state number 7.”

Such generalizations are made more easily by using the alignment information generated in the first step of the training process. Each arc is marked when it is created with the index of the output phoneme (if any) to which the input phoneme corresponds in the output string. During the merging step of the algorithm, the merging of otherwise similar arcs with different stored indices is disallowed. (In practice, the other constraints generally prevent this from happening anyway.)

For each state, the arcs leaving the state were classified into groups that agree on each of the following features:

- the index i of the output symbol corresponding to the input symbol
- the difference of the phonetic feature vectors of the input symbol and symbol i of the output string
- the prefix of length $i - 1$ of the output string
- the suffix of the output string beginning at position $i + 1$

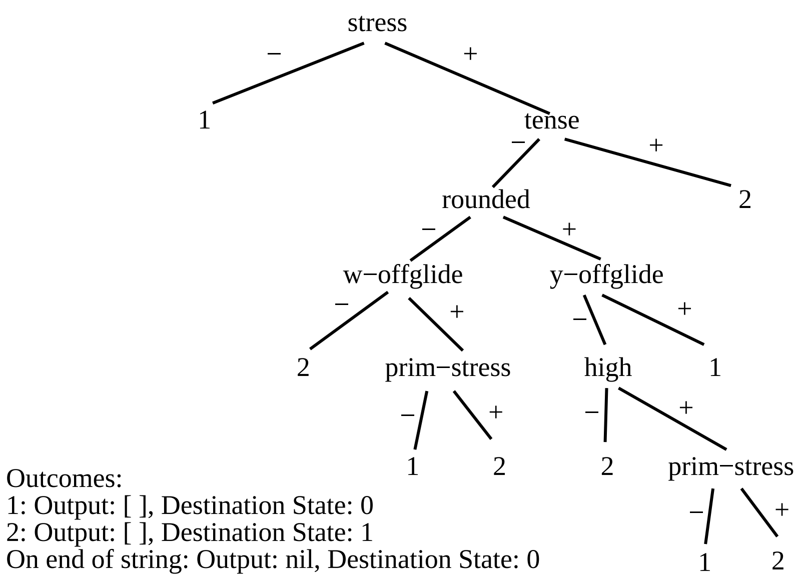
After the process of merging states terminates, a decision tree is induced at each state to classify the outgoing arcs. The branches of the decision tree are labeled with phonetic feature values of the arc’s input symbol, and the leaves of the tree correspond to the groups described above. The same 26 binary phonetic features used in calculating edit distance was used to classify phonemes in the decision trees. Arcs whose input is the end of string symbol are not included in the decision trees, but rather simply left as is.

Using phonetic features to build a decision tree guarantees that each leaf of the tree represents a natural class of phonemes, that is, a set of phonemes that can be described by specifying values for some subset of the phonetic features. Thus if we think of the transducer as a set of rewrite rules, the decision tree expresses the preceding context as a regular expression of natural classes of preceding phonemes. Because these classes are expressed in terms of phonetic features, the transducers augmented with the decision trees will generalize to new contexts which share phonetic features with the training contexts.

Some induced transducers may need to be generalized even further, since the input transducer to the decision tree learning may have arcs which are incorrect merely because of accidental prior structure. Consider again the English flapping rule, which applies in the context of a preceding stressed vowel. Our algorithm first learned a transducer whose decision tree is shown in Figure 12. In this transducer all arcs leaving state 0 correctly lead to the flapping state on stressed vowels, except for those stressed vowels which happen not to have occurred in the training set. For these unseen vowels (which consisted of the rounded diphthongs ‘oy’ and ‘ow’ with secondary stress), the transducers incorrectly returns to state 0. In this case, we wish the algorithm to make the generalization that the rule applies after *all* stressed vowels.

This type of generalization can be accomplished by pruning the decision trees at each state of the machine. Pruning is done by stepping through each state of the machine and pruning as many decision nodes as possible at each state. The entire training set of transductions is tested after each branch is pruned. If any errors are found, the outcome of the pruned node’s other child is tested. If errors are still found, the pruning operation is reversed. This process continues at the fringe of the decision tree until no more pruning is possible. Figure 13 shows the correct decision tree for flapping after pruning.

The process of pruning the decision trees is complicated by the fact that the pruning operations allowed at one state depend on the status of the trees at each other state. Thus it is necessary to make several passes through the states, attempting additional pruning at each pass, until no more improvement is possible. In addition, testing each pruning operation against the entire training set is expensive, but in the case of synthetic data it gives the best results. For other applications it may be desirable to keep a cross validation set for this purpose.



Outcomes:
1: Output: [ ], Destination State: 0
2: Output: [ ], Destination State: 1
On end of string: Output: nil, Destination State: 0

Figure 12: *Decision Tree Before Pruning*: The initial state of the flapping transducer

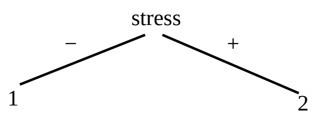


Figure 13: *The Same Decision Tree After Pruning*

7 Results and Discussion

We tested our induction algorithm on a synthetic corpus of 49,280 input/output pairs. Each pair consisted of an underlying and a surface pronunciation of an individual word of English. The underlying string of each pair was taken from the phoneme-based CMU pronunciation dictionary. The surface string was generated from each underlying form by mechanically applying the one or more rules we were attempting to induce in each experiment.

In our first experiment, we applied the flapping rule in (2) to training corpora of between 6250 and 50,000 words. Figure 14 shows the transducer induced from 50,000 training samples, and Figure 15 shows some performance results.

(2) $t \rightarrow dx / \acute{V} r^* \_\_ V$

As can be seen from Figure 15, the use of alignment information in creating the initial tree transducer dramatically decreases the number of states in the learned transducer and the performance on test data. The improved algorithm induced a flapping transducer with the minimum number of states with as few as 6250 samples. The use of alignment information also reduced the learning time; the additional cost of calculating alignments is more than compensated for by quicker merging.

The algorithm also successfully induced transducers with the minimum number of states for the t-insertion and t-deletion rules below, given only 6250 samples.

For the r-deletion rule in (3), the algorithm induced a machine which was not the theoretical minimal machine, as Figure 16 shows. We discuss these results below.

(3) $r \rightarrow \emptyset / [+vocalic] \_\_ [+consonantal]$

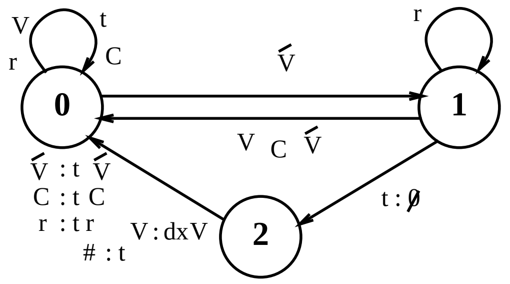


Figure 14: *Flapping Transducer Induced from 50,000 Samples*

	OSTIA w/o Alignment		OSTIA w/ Alignment	
Samples	States	% Error	States	% Error
6250	19	2.32	3	0.34
12500	257	16.40	3	0.14
25000	141	4.46	3	0.06
50000	192	3.14	3	0.01

Figure 15: *Results Using Alignment Information on English Flapping*

In our second experiment, we applied our learning algorithm to a more difficult problem: inducing multiple rules at once. Setting r-deletion aside for present, a data set was constructed by applying the t-insertion rule in (4), the t-deletion rule in (5) and the flapping rule already seen in (2) one after another. As these rules do not affect one another's environment, the order of their application is not significant. The minimum number of states for a subsequential transducer performing the composition of the three rules is five. As is seen in Figure 17, a transducer of minimum size was obtained with 12500 or more sample transductions.

(4) $\emptyset \rightarrow t / n\_\_ s$

(5) $t \rightarrow \emptyset / n\_\_ \begin{bmatrix} +vocalic \\ -stress \end{bmatrix}$

The effects of adding decision tress at each state of the machine for the composition of t-insertion, t-deletion and flapping are shown in Figure 18. By making it impossible to fall of the transducer, errors are reduced by about 80%. Pruning the decision trees further reduces errors to under one in 10000.

Figure 19 shows the final transducer induced from this corpus of 12,500 words with pruned decision trees.

An analysis of errors in the induction suggests three separate ways in which the induction algorithm could be improved. Our first problem was the difficulty of inducing a transducer for r-deletion. The problem was not deletion per se, since our algorithm successfully learns the t-deletion rule. Rather, we believe that the difficulty with r-deletion is the broad context in which the rule applies: after any vowel and before any consonant. Since our phoneme set distinguishes three degrees of stress for each vowel, the alphabet size is 72;

	R-deletion	
Samples	States	% Error
6250	4	0.48
12500	3	0.21
25000	6	0.18
50000	35	0.30

Figure 16: *Results on R-deletion using Alignment Information*

	OSTIA w/Alignment	
Samples	States	% Error
6250	6	0.93
12500	5	0.20
25000	5	0.09
50000	5	0.04

Figure 17: *Results on Three Rules Composed*

Method	States	% Error
OSTIA	329	22.09
Alignment	5	0.20
Add D-trees	5	0.04
Prune D-trees	5	0.01

Figure 18: *Results on Three Rules Composed* 12,500 Training, 49,280 Test

we believe this was simply too large for the algorithm without some prior concept of ‘vowel’ and ‘consonant’. While our decision tree augmentation adds these concepts to the algorithm, it only does so after the initial transducer has been induced, and so cannot help in building the initial transducer. We need some method of interleaving the generalization of phonemes into classes, performed by the decision trees, and the induction of the structure of the transducer by merging states. Making generalizations about input phonemes would in effect reduce the alphabet size on the fly, making structure-learning easier.

An examination of the few errors (three samples) in the induced flapping and three-rule transducers points out another flaw in our model. While the learned transducer correctly makes the generalization that flapping occurs after any stressed vowel, it does not flap after two stressed vowels in a row. Upon seeing a stressed vowel at the initial state, a transition is made to the state from which flapping can occur. However, from this state, seeing another stressed vowel causes a transition back to the initial state. This is possible because no samples containing two stressed vowels in a row (or separated by an ’r’) immediately followed by a flap were in the training data.

This transducer will thus flap a ’t’ after any odd number of stressed vowels, rather than simply after any stressed vowel. Such a rule seems quite unnatural phonologically, and makes for an odd context-sensitive rewrite rule. Any sort of simplest hypothesis criterion applied to a system of rewrite rules would prefer a rule such as

$$t \rightarrow dx / \acute{V} \_\_ \; V$$

to a rule such as

$$t \rightarrow dx / \acute{V}(\acute{V}\acute{V})^{*} \_\_ \; V$$

which is the equivalent of the transducer learned from the training data. Such a rule, however, is perfectly natural for a transducer. This suggests that, although the traditional formalism of context-sensitive rewrite rules may be no more powerful than finite transducers, it contains implicit generalizations about how phonological rules usually work that are not present in the transducer system. We hope that further experimentation will lead to a way of expressing this language bias in our induction system.

Finally, subsequential transducers are an inefficient representation of some sorts of rules. While all the rules discussed above can be represented with transducers of two or three states, rules applying to an entire class of phonemes can lead to an explosion in the number of necessary states. This is because the transducer must wait to see the right hand context of a rule before emitting the rule’s output, and must therefore remember what that output is to be. One example is word-final devoicing of obstruents:

$$\left[\; +obstruent \;\right] \rightarrow \left[\; -voiced \;\right] / \_\_ \; \#$$

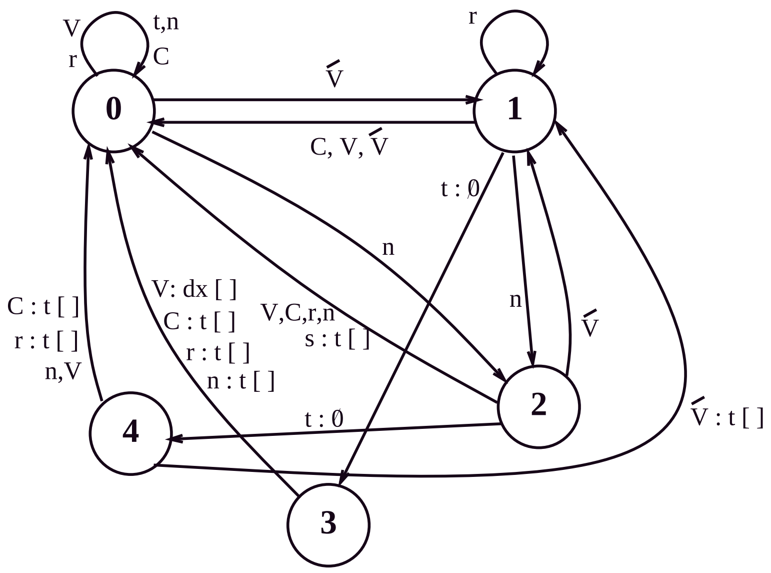


Figure 19: *Three Rule Transducer Induced from 12,500 Samples*

In this case, a separate state must be created for each obstruent subject to devoicing. The transducer would jump to the one of these states, without emitting any output, upon seeing the appropriate obstruent. Upon seeing the end of word symbol at this state, the corresponding unvoiced obstruent will be emitted. Upon seeing any other next symbol, the original voiced obstruent will be emitted.

One way around this would to add a memory to the model. The transducer could keep track of the input symbols seen so far. Just as the generalized arcs can now specify one of their output symbols as being the input symbol with certain phonetic features changed, they would be able to reference previous input symbols. This method would achieve best results if used with an algorithm that interleaves the merging of states and generalizations of arcs, so that a transducer with a large number of states would not have to be first correctly learned before merging states by using the memory mechanism.

8 Related Work

Recent work in the machine learning of phonology includes algorithms for learning both segmental and non-segmental information. Non-segmental approaches include those of Daelemans *et al.* (1994) for learning stress systems, as well as approaches to learning morphology such as Gasser's (1993) system for inducing Semitic morphology, and Ellison's (1992) extensive work on syllabicity, sonority, and harmony. Since our approach learns solely segmental structure, a more relevant comparison is with other algorithms for inducing segmental structure; this includes SPE phonological rules as well as modern autosegmental association rules.

Johnson (1984) gives one of the first computational algorithms for phonological rule induction. His algorithm works for rules of the form

(6) $a \rightarrow b/C$

where C is the feature matrix of the segments around a. Johnson's algorithm sets up a system of constraint equations which C must satisfy, by considering both the positive contexts, i.e., all the contexts C_i in which a b occurs on the surface, as well as all the negative contexts C_j in which an a occurs on the surface. The set of all positive and negative contexts will not generally determine a unique rule, but will determine a set of possible rules. Johnson then proposes that principles from universal grammar might be used to choose between candidate rules, although he does not suggest any particular principles.

Johnson's system, while embodying an important insight about the use of positive and negative contexts for learning, did not generalize to insertion and deletion rules, and it is not clear how to extend his system to modern autosegmental phonological systems. Touretzky *et al.* (1990) extended Johnson's insight by using the *version spaces* algorithm of Mitchell (1981) to induce phonological rules in their *Many Maps* architecture. Rules in their architecture resemble a three-level version of the two-level rules of Koskenniemi (1983). Like Johnson's, their system looks at the underlying and surface realizations of single segments. For each segment, the system uses the version space algorithm to search for the proper statement of the context. The model also has a separate algorithm which handles harmonic effects by looking for multiple segmental changes in the same word, and has separate processes to deal with epenthesis and deletion rules. Touretzky *et al.*'s approach seems quite promising; our use of decision trees to generalize each state is a similar use of phonological feature information to form generalizations. We hope that in making our generalization operator more on-line, we can make use of some of the negative context evidence that helps Johnson's and Touretzky *et al.*'s systems converge.

Riley (1991) and Withgott & Chen (1993) propose a decision-tree approach to segmental mapping. A decision tree is induced for each phoneme, classifying possible realizations of the phoneme in terms of contextual factors such as stress and the surrounding phonemes. The technique's major advantages are its probabilistic and data-driven nature, as well as its ability to generalize based on phonological information. The disadvantage of the decision-tree technique by itself is that it misses generalizations about the behavior of similar phonemes; the decision tree for each phoneme is learned separately. In addition, no generalizations are made about similar context phonemes. In a transducer based formalism, generalizations about similar context phonemes naturally follow from generalizations about individual phonemes' behavior, as the context is represented by the current state of the machine, which in turn depends on the behavior of the machine on the previous phonemes.

The decision tree framework also makes long distance dependencies harder to learn than does a transducer-based framework. To model rules with more distant contexts, such as vowel harmony rules, one must add more distant phonemes to the features used to learn the decision tree. This complicates the learning process and makes the resulting trees unwieldy. To represent such a vowel harmony rule, a transducer can enter a new state upon seeing the trigger for the harmony and remain in this state, or a set of related states, for as long as the effects of the harmony last.

9 Conclusion

Inferring finite state transducers seems to hold promise as a method for learning phonological rules. Both of our initial augmentations of OSTIA to bias it toward phonological naturalness improve performance. Using information on the alignment between input and output strings allows the algorithm to learn more compact, more accurate transducers. The addition of decision trees at each state of the resulting transducer further improves accuracy and results in phonologically more natural transducers. We believe that further and more integrated uses of phonological naturalness, such as generalizing across similar phenomena at different states of the transducer, interleaving the merging of states and generalization of transitions, and adding memory to the model of transduction, could help even more.

Our current algorithm and most previous algorithms are designed for obligatory rules. These algorithms fail completely when faced with optional, probabilistic rules, such as flapping. This is the advantage of probabilistic approaches such as the Riley/Withgott approach. One area we hope to investigate is the generalization of our algorithm to probabilistic rules with probabilistic finite-state transducers, perhaps by augmenting PFST induction techniques such as Stolcke & Omohundro (1994) with insights from phonological naturalness.

Besides aiding in the development of a practical tool for learning phonological rules, our results point to the use of constraints from universal grammar as a strong factor in the machine and possibly human learning of natural language phonology.

Acknowledgments

Thanks to Jerry Feldman, Isabel Galiano-Ronda, Jose Oncina, Andreas Stolcke, Gary Tajchman, Lauri Karttunen, and Eric Fosler. This work was partially funded by ICSI.

References

DAELEMANS, WALTER, STEVEN GILLIS, & GERT DURIEUX. 1994. The acquisition of stress: A data-oriented approach. *Computational Linguistics* 208.421–451.

ELLISON, T. MARK, 1992. *The Machine Learning of Phonological Structure*. University of Western Australia dissertation.

GASSER, MICHAEL, 1993. Learning words in time: Towards a modular connectionist account of the acquisition of receptive morphology. Draft.

JOHNSON, C. DOUGLAS. 1972. *Formal Aspects of Phonological Description*. The Hague: Mouton.

JOHNSON, MARK. 1984. A discovery procedure for certain phonological rules. In *Proceedings of the Tenth International Conference on Computational Linguistics*, 344–347, Stanford.

KAPLAN, RONALD M., & MARTIN KAY. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20.331–378.

KARTTUNEN, LAURI. 1993. Finite-state constraints. In *The Last Phonological Rule*, ed. by John Goldsmith. University of Chicago Press.

KOSKENNIEMI, KIMMO. 1983. Two-level morphology: A general computational model of word-form recognition and production. *Publication No. 11, Department of General Linguistics, Univ of Helsinki* .

MITCHELL, TOM M. 1981. Generalization as search. In *Readings in Artificial Intelligence*, ed. by Bonnie Lynn Webber & Nils J. Nilsson, 517–542. Los Altos: Morgan Kaufmann.

ONCINA, JOSÉ, PEDRO GARCÍA, & ENRIQUE VIDAL. 1993. Learning subsequential transducers for pattern recognition tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.448–458.

RILEY, MICHAEL D. 1991. A statistical model for generating pronunciation networks. In *IEEE ICASSP-91*, 737–740.

STOLCKE, ANDREAS, & STEPHEN OMOHUNDRO. 1994. Best-first model merging for hidden Markov model induction. Technical Report TR-94-003, International Computer Science Institute, Berkeley, CA.

TOURETZKY, DAVID S., GILLETTE ELVGREN III, & DEIRDRE W. WHEELER. 1990. Phonological rule induction: An architectural solution. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society (COGSCI-90)*, 348–355.

WAGNER, R. A., & M. J. FISCHER. 1974. The string-to-string correction problem. *Journal of the Association for Computation Machinery* 21.168–173.

WITHGOTT, M. M., & F. R. CHEN. 1993. *Computation Models of American Speech*. Center for the Study of Language and Information.