To appear in ACL-95

Automatic Induction of Finite State Transducers for Simple Phonological Rules

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

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, biases from "universal grammar". One bias is that underlying phones are often realized as phonetically similar or identical surface phones. The other biases phonological rules to apply across natural phonological classes. 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.

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. This means that finite state transducers can be used to represent phonological rules, greatly simplifying 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 for learning rules from sample pairs of input and output strings, and the results we obtained.

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) t → dx / $\acute{V}$ r* ___ V

2 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 to provide background for the modifications that follow.

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. The root of the tree is the transducer's initial state, and each leaf of the tree corresponds to the end of an input sample.

The output symbols are placed as near the root of the tree as possible while avoiding conflicts in the output of a given arc. An example of the result of this initial tree construction is shown in Figure 2.

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

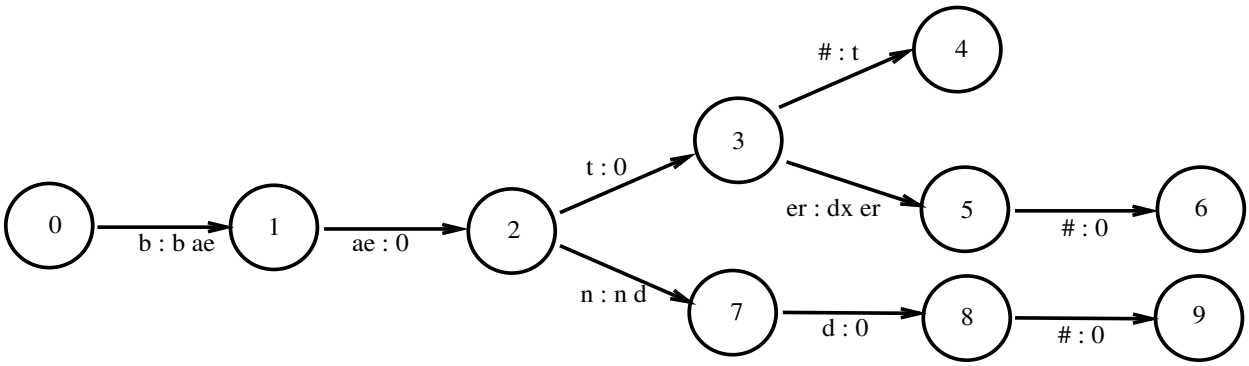


Figure 2: *Onward Tree Transducer for "bat", "batter", and "band" with Flapping Applied*

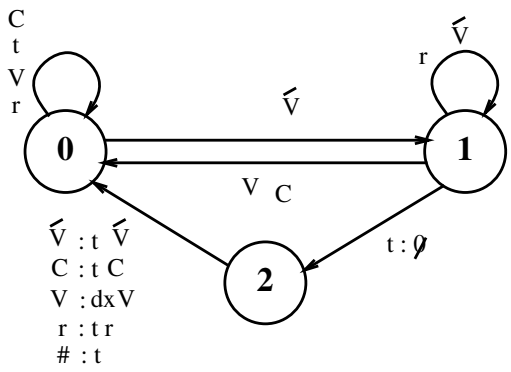


Figure 1: *Subsequential Transducer for English Flapping*: Labels on arcs are of the form (input symbol):(output symbol). Labels with no colon indicate the same input and output symbols. 'V' indicates any unstressed vowel, 'V́' any stressed vowel, 'dx' a flap, and 'C' any consonant other than 't', 'r'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

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 2 is shown in Figure 3, and the end result of the OSTIA alogrithm in shown in Figure 4.

3 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

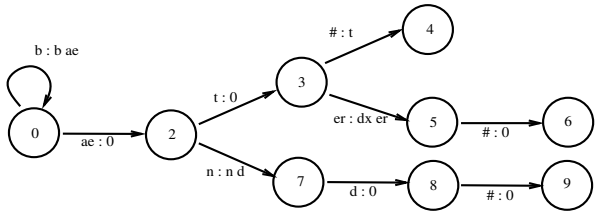


Figure 3: *Result of Merging States 0 and 1 of Figure 2*

far. However, when trying to learn phonological rules from linguistic data, the necessary training set may not be available. In particular, systematic phonological constraints such as syllable structure may rule out the necessary strings. The algorithm does not have the language bias which would allow it to avoid linguistically unnatural transducers.

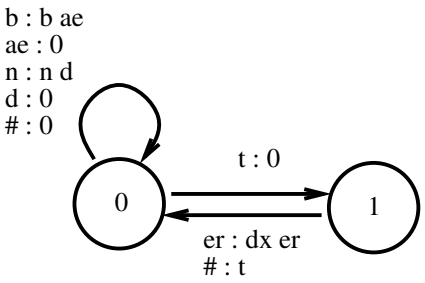


Figure 4: *Final Result of Merging Process on Transducer from Figure 2*

For example, OSTIA's tendency to produce overly "clumped" transducers is illustrated by the arcs with out "b ae" and "n d" in the transducer in Figure 4, or even Figure 2. OSTIA's 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. This 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.

Another example of an unnatural generalization is shown in 4, the final transducer induced by OSTIA on the three word training set of Figure 2. For example, the transducer of Figure 4 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 end-of-string after a 't'. While it might be unreasonable to expect any transducer trained on three samples to be perfect, the transducer of Figure 4 illustrates on a small scale how the OSTIA algorithm might be improved.

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 generalize the environment to all stressed vowels. The algorithm needs knowledge about classes of phonemes to fill in accidental gaps in training data coverage.

4 Using Alignment Information

Our first modification of OSTIA was to add the bias that, as a default, a phoneme is realized as itself, or as a similar phone. Our algorithm guesses the most probable phoneme to phoneme alignment between the input and output strings, and uses this information to distribute the output symbols among the arcs of the initial tree transducer. This is demonstrated for the word "importance" in Figures 5 and 6.

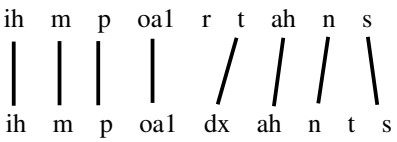


Figure 5: Alignment of "importance" with flapping, r-deletion and t-insertion

The modification proceeds in two stages. First, a dynamic programming method is used to compute a correspondence between input and output phonemes. 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 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 7, for comparison with the unaligned version in Figure 2.

Results of our alignment algorithm are summarized in §6. The denser distribution of output symbols resulting from the alignment constrains the merging of states early in the merging loop of the algorithm. Interestingly, preventing the wrong states from merging early on allows more merging later, and results in more compact transducers.

5 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 8. Note that if the underlying phone is an unstressed vowel ([-cons,-stress]), the machine outputs a flap, followed by the underlying vowel, otherwise it outputs a 't' followed by the underlying phone.

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. The decision trees classify the arcs leaving each state based on the arc's input symbol into groups with the same behavior. The same 26 binary phonetic features used in calculating edit distance were used to classify phonemes in the decision trees. Thus 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 different behaviors. By an arc's behavior, we mean its output string considered as a function of its input phoneme, and its destination state. Two arcs are considered to have the same behavior if they agree each of the following:

- the index i of the output symbol corresponding to the input symbol (determined from the alignment procedure)
- 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

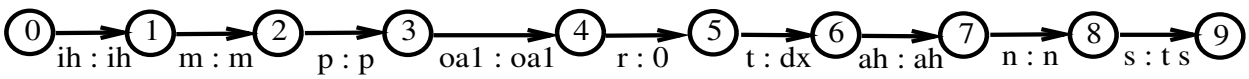


Figure 6: Resulting initial transducer for "importance"

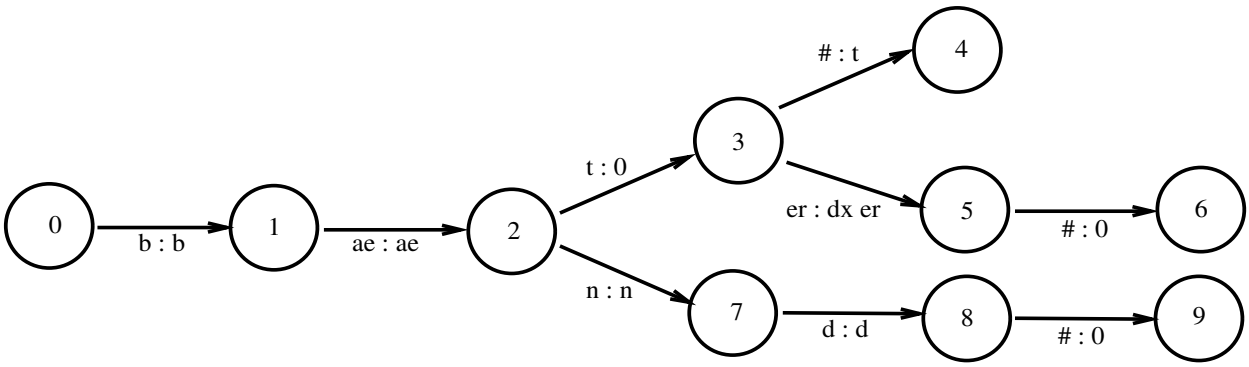
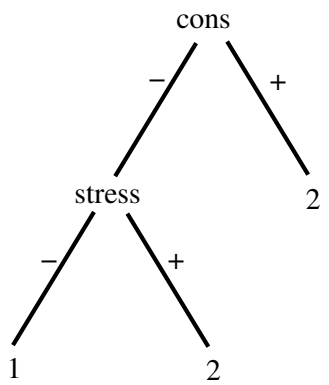


Figure 7: *Initial Tree Transducer Constructed with Alignment Information*: Note that output symbols have been pushed back across state 3 during the construction



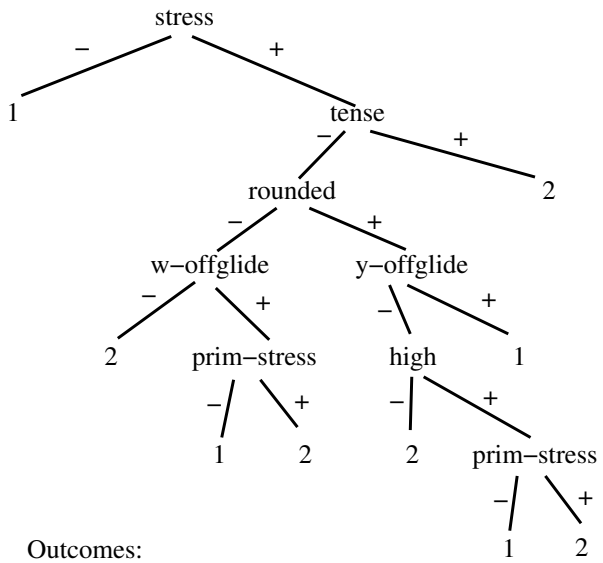
Outcomes:
1: Output: dx [ ], Destination State: 0
2: Output: t [ ], Destination State: 0
3: On end of string: Output: t, Destination State: 0

Figure 8: *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 suffix of the output string beginning at position $i + 1$
- the destination state

After the process of merging states terminates, a decision tree is induced at each state to classify the outgoing arcs. Figure 9 shows a tree induced at the initial state of the transducer for flapping.

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, we can now express the context of each rule as a regular expression of natural classes of preceding phonemes.



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

Figure 9: *Decision Tree Before Pruning*: The initial state of the flapping transducer

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 9. 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.

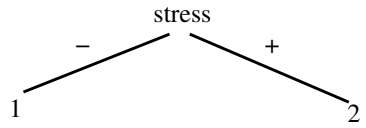


Figure 10: *The Same Decision Tree After Pruning*

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 10 shows the correct decision tree for flapping, obtained by pruning the tree in Figure 9.

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.

6 Results and Discussion

We tested our induction algorithm using a synthetic corpus of 99,279 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 11 shows the transducer induced from 50,000 training samples, and Figure 12 shows some performance results.

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

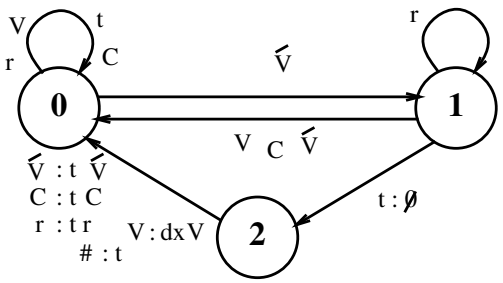


Figure 11: *Flapping Transducer Induced from 50,000 Samples*

	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 12: *Results Using Alignment Information on English Flapping*

As can be seen from Figure 12, the use of alignment information in creating the initial tree transducer dramatically decreases the number of states in the learned transducer as well as the error 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 of states.

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.

In our second experiment, we applied our learning algorithm to a more difficult problem: inducing multiple rules at once. A data set was constructed by applying the t-insertion rule in (3), the t-deletion rule in (4) and the flapping rule already seen in (2) one after another. As is seen in Figure 13, a transducer of minimum size (five states) was obtained with 12500 or more sample transductions.

(3) $\emptyset \rightarrow t / n __ s$

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

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 14.

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

Figure 13: *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 14: *Results on Three Rules Composed* 12,500 Training, 49,280 Test

Figure 15 shows the final transducer induced from this corpus of 12,500 words with pruned decision trees.

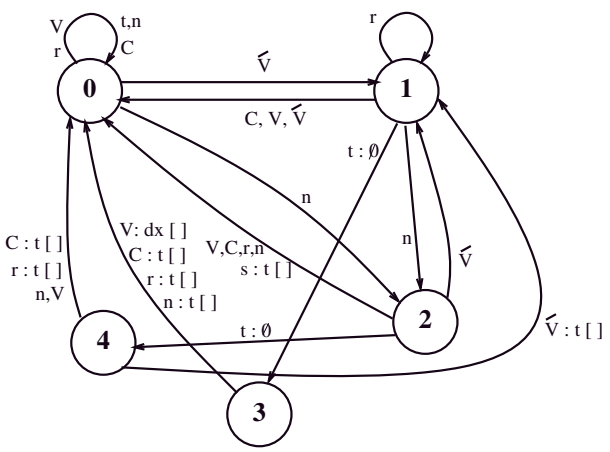


Figure 15: *Three Rule Transducer Induced from 12,500 Samples*

An examination of the few errors (three samples) in the induced flapping and three-rule transducers points out a 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. 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 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. This suggests that, the traditional formalism of context-sensitive rewrite rules 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.

7 Related Work

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

(5) $\quad a \rightarrow b / C$

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.

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. 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.

Riley (1991) and Withgott & Chen (1993) first proposed 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. However, since the decision tree for each phoneme is learned separately, the the technique misses generalizations about the behavior of similar phonemes. 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.

We hope that our hybrid model will be more successful at learning long distance dependencies than the simple decision tree approach. To model long distance rules such as vowel harmony in a simple decision tree approach, one must add more distant phonemes to the features used to learn the decision tree. In a transducer, this information is represented in the current state of the transducer.

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

References

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.

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

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.