Learning String Edit Distance[1]

Eric Sven Ristad Peter N. Yianilos

Research Report CS-TR-532-96
October 1996

Abstract

In many applications, it is necessary to determine the similarity of two strings. A widely-used notion of string similarity is the edit distance: the minimum number of insertions, deletions, and substitutions required to transform one string into the other. In this report, we provide a stochastic model for string edit distance. Our stochastic model allows us to learn the optimal string edit distance function from a corpus of examples. We illustrate the utility of our approach by applying it to the difficult problem of learning the pronunciation of words in conversational speech. In this application, we learn a string edit distance function with one third the error rate of the untrained Levenshtein distance.

Keywords: string edit distance, Levenshtein distance, stochastic transduction, pronunciation recognition, spelling correction, string correction, speech recognition, Switchboard corpus.

[1]Both authors are with the Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08544. The second author is also with the NEC Research Institute, 4 Independence Way, Princeton, NJ 08540. The first author is partially supported by Young Investigator Award IRI-0258517 from the National Science Foundation. Email: {ristad,pny}@cs.princeton.edu.

1 Introduction

In many applications, it is necessary to determine the similarity of two strings. A widely-used notion of string similarity is the edit distance: the minimum number of insertions, deletions, and substitutions required to transform one string into the other [10]. In this report, we provide a stochastic model for string edit distance. Our stochastic interpretation allows us to learn the optimal string edit distance function from a corpus of examples. It also leads to a variant of string edit distance, that aggregates the many different ways to transform one string into another. We illustrate the utility of our approach by applying it to the difficult problem of learning the pronunciation of words in the Switchboard corpus of conversational speech [5]. In this application, we learn a string edit distance function that reduces the error rate of the untrained Levenshtein distance by a factor of three.

Let us first define our notation. Let A be a finite alphabet of distinct symbols and let $x^T \in A^T$ denote an arbitrary string of length T over the alphabet A. Then x_i^j denotes the substring of x^T that begins at position i and ends at position j. For convenience, we abbreviate the unit length substring x_i^i as x_i and the length t prefix of x^T as x^t.

A string edit distance function is characterized by a triple $\langle A, B, c \rangle$ consisting of the finite alphabets A and B and the primitive cost function $c : E \rightarrow \Re_+$ where $\Re_+$ is the set of nonnegative reals, $E = E_s \cup E_d \cup E_i$ is the alphabet of primitive edit operations, $E_s = A \times B$ is the set of the substitutions, $E_d = A \times \{\epsilon\}$ is the set of the deletions, and $E_i = \{\epsilon\} \times B$ is the set of the insertions. Each such triple $\langle A, B, c \rangle$ induces a distance function $d_c : A^* \times B^* \rightarrow \Re_+$ that maps a pair of strings to a nonnegative value. The distance $d_c(x^t, y^v)$ between two strings $x^t \in A^t$ and $y^v \in B^v$ is defined recursively as

$$d_c(x^t, y^v) = \min \left\{ \begin{array}{l} c(x_t, y_v) + d(x^{t-1}, y^{v-1}), \\ c(x_t, \epsilon) + d(x^{t-1}, y^v), \\ c(\epsilon, y_v) + d(x^t, y^{v-1}) \end{array} \right\} \tag{1}$$

where $d_c(\epsilon, \epsilon) = 0$. The edit distance may be computed efficiently using dynamic programming [12, 18]. Many excellent reviews of the string edit distance literature are available [7, 9, 14, 17]. Several variants have been proposed, including the constrained edit distance [13] and the normalized edit distance [11].

A stochastic interpretation of string edit distance was first noted by Hall and Dowling [7, p.390-1] in their review of approximate string matching algorithms, but without a proposal for learning the edit costs. The principal contribution of this report is an effective algorithm for learning the primitive edit costs. To the best of our knowledge, this is the first published algorithm for learning the primitive edit costs. We first implemented our approach in August 1993 for the problem of classifying greyscale images of handwritten digits.

The remainder of this report consists of two sections and two appendices. In section 2, we define our stochastic model of string edit distance and provide an

efficient algorithm to learn the primitive edit costs from a corpus of string pairs. In section 3, we apply our approach to the difficult problem of learning the pronunciations of words in conversational speech. In appendix A, we present additional results for the pronunciation recognition problem. In appendix B, we present an alternate model of string edit distance where probabilities are conditioned on string lengths.

2 A Stochastic Model

We model string edit distance as a memoryless stochastic transduction from the underlying strings A^* to the surface strings B^*. Each step of the transduction generates either a substitution pair $\langle a, b \rangle$, a deletion pair $\langle a, \epsilon \rangle$, or an insertion pair $\langle \epsilon, b \rangle$ according to a probability function $\delta : E \rightarrow [0, 1]$. Being a probability function, $\delta(\cdot)$ satisfies the following constraints:

$$\begin{array}{ll} a. & \forall z \in E \; [\; 0 \leq \delta(z) \leq 1 \;] \\ b. & \sum_{z \in E} \delta(z) = 1 \end{array}$$

Note that the null operation $\langle \epsilon, \epsilon \rangle$ is not included in the alphabet E of edit operations.

In appendix B, we consider an alternate parameterization of the memoryless stochastic transducer that more cleanly supports transductions conditioned on string lengths. In the alternate parameterization, the transition probability $\delta(\cdot)$ is represented the product of the probability of choosing the type of edit operation (insertion, deletion, or substitution) and the conditional probability of choosing the symbol(s) used in the edit operation.

2.1 Generation

A memoryless stochastic transducer $\phi = \langle A, B, \delta \rangle$ naturally induces a probability function $p(\cdot|\phi, n)$ on the space E^n of all edit sequences of a given length n. This probability function is defined by the following generation algorithm.

GENERATE(ϕ, n)
1. For $i = 1$ to n
2. Pick z_i from E according to $\delta(\cdot)$
3. return(z^n);

In our intended applications, we require a probability function on string pairs rather than on edit sequences. In order to obtain such a probability function, we proceed as follows. First, we let $p(\cdot|\phi)$ be a probability function on any maximal prefix-free subset of E^*, that is, any maximal set of edit sequences such that no edit sequence in the set is a proper prefix of another edit sequence in the set.

Next, we let $\nu(z^n) \in A^* \times B^*$, be the *yield* of the edit sequence z^n. Then we define $p(x^T, y^V|\phi)$ to be the marginal of the joint probability $p(x^T, y^V, z^n|\phi)$ where $p(x^T, y^V, z^n|\phi) = p(z^n|\phi)$ if $\nu(z^n) = \langle x^T, y^V \rangle$, and otherwise zero. Therefore,

$$\begin{aligned} p(x^T, y^V|\phi) &= \sum_{z^n \in E^*} p(x^T, y^V, z^n|\phi) \\ &= \sum_{\{z^n : \nu(z^n) = \langle x^T, y^V \rangle\}} p(z^n|\phi) \end{aligned} \tag{2}$$

where the probability $p(z^n|\phi, n)$ of an edit sequence $z^n \in E^n$ is simply the product of the probabilities $\delta(z_i)$ of the individual edit operations because the transducer is memoryless. The resulting function $p(\cdot, \cdot|\phi)$ is a probability function on any maximal prefix-free set of string pairs.[2] In appendix B, we define a related probability function on all sets of string pairs of a given length.

2.2 Two Distances

Our interpretation of string edit distance as a memoryless stochastic transduction leads to the following two string distance functions. The first distance $d^v_\phi(\cdot, \cdot)$ is defined by the most likely memoryless transduction between the two strings while the second distance $d^s_\phi(\cdot, \cdot)$ is defined by agreggating all memoryless transductions between the two strings.

The first transduction distance $d^v_\phi(x^T, y^V)$, which we call the *Viterbi edit distance*, is the negative logarithm of the probability of the most likely edit sequence for the string pair $\langle x^T, y^V \rangle$

$$d^v_\phi(x^T, y^V) \doteq -\log \mathsf{argmax}_{\{z^n : \nu(z^n) = \langle x^T, y^V \rangle\}} \{p(z^n|\phi)\} \tag{3}$$

This distance function is identical to the string edit distance $d_c(\cdot, \cdot)$ where the edit costs are set to the negative logarithm of the edit probabilities, that is, where $c(z) \doteq -\log \delta(z)$ for all $z \in E$.

The second transduction distance $d^s_\phi(x^T, y^V)$, which we call the *stochastic edit distance*, is the negative logarithm of the total probability of the prefix pair $\langle x^T, y^V \rangle$.

$$d^s_\phi(x^T, y^V) \doteq -\log p(x^T, y^V|\phi) \tag{4}$$

This second distance function differs from the first in that it considers the contribution of all ways to simultaneously generate the two strings. If the most likely edit sequence for $\langle x^T, y^V \rangle$ is significantly more likely than any of the other edit sequences, then the two transduction distances will be nearly equal. However, if a given string pair has many likely generation paths, then the stochastic distance $d^s_\phi(\cdot, \cdot)$ will be less than the Viterbi distance $d^v_\phi(\cdot, \cdot)$.

Unlike the classic edit distance $d_c(\phi, \phi)$, our two transduction distances are never zero unless they are infinite for all other string pairs in a prefix-free set

[2] Note that $p(\cdot, \cdot|\phi)$ is not a probability over finite strings of fixed lengths because the sum of $p(x^t, y^v|\phi)$ over all finite strings $x^t \in A^t$ and $y^v \in B^v$ of fixed lengths t and v will in general be less than unity.

of string pairs. Recall that the Levenshtein distance assigns zero cost to all identity edit operations. Therefore, an infinite number of identity edits is less costly than even a single insert, delete, or substitute. The only way to obtain this property in a transduction distance is to assign zero probability (ie., infinite cost) to all nonidentity operations, which would assign finite distance only to pairs of identical strings.

2.3 Evaluation

Our generative model assigns probability to edit sequences and the string pairs that they yield. Each pair of strings may be generated by a great many different edit sequences. Therefore we must calculate the marginal probability $p(x^T, y^V|\phi)$ of a pair of strings by summing the joint probability $p(x^T, y^V, z^n|\phi)$ over all the edit sequences that yield the given string pairs (2).

Each string pair is generated by exponentially many edit sequences, and so it would not be feasible to evaluate the probability of a string pair by actually summing over all its edit sequences (2). Fortunately, the following dynamic programming algorithm calculates the probability $p(x^T, y^V|\phi)$ in $O(T \cdot V)$ time and space. At the end of the computation, the $\alpha_{t,v}$ entry contains the probability $p(x^t, y^v|\phi)$ of the prefix pair $\langle x^t, y^v \rangle$ and $\alpha_{T,V}$ is the probability of the entire string pair.

FORWARD-EVALUATE(x^T,y^V,ϕ)
1. $\alpha_{0,0} := 1$;
2. For $t = 0 \ldots T$
3. For $v = 0 \ldots V$
4. if $(v > 1 \vee t > 1)$ [$\alpha_{t,v} := 0$;]
5. if $(v > 1)$ [$\alpha_{t,v}$ += $\delta(\epsilon, y_v)\alpha_{t,v-1}$;]
6. if $(t > 1)$ [$\alpha_{t,v}$ += $\delta(x_t, \epsilon)\alpha_{t-1,v}$;]
7. if $(v > 1 \wedge t > 1)$ [$\alpha_{t,v}$ += $\delta(x_t, y_v)\alpha_{t-1,v-1}$;]
8. return(α);

2.4 Estimation

Under our stochastic model of string edit distance, the problem of learning the edit costs reduces to the problem of estimating the parameters of a memoryless stochastic transducer. For this task, we employ the powerful expectation maximization (EM) algorithm [2, 3, 4]. The EM algorithm is an iterative algorithm that maximizes the probability of the training data according to the model. See [15] for a review. As its name suggests, the EM algorithm consists of two steps. In the expectation step, we accumulate the expectation of each hidden event on the training corpus. In our case the hidden events are the edit operations used to generate the string pairs. In the maximization step, we set our parameter values to their relative expectations on the training corpus.

The following EXPECTATION-MAXIMIZATION() algorithm optimizes the parameters ϕ of a memoryless stochastic transducer on a corpus $C = \langle x^{T_1}, y^{V_1} \rangle$, ..., $\langle x^{T_n}, y^{V_n} \rangle$ of n training pairs. Each iteration of the EM algorithm is guaranteed to either increase the probability of the training corpus or not change the model parameters. The correctness of our algorithm is shown in related work [16].

```
EXPECTATION-MAXIMIZATION(φ, C)
1.  until convergence
2.      forall z in E [ γ(z) := 0; ]
3.      for i = 1 to n
4.          EXPECTATION-STEP(x^{T_i}, y^{V_i}, φ, γ);
5.      MAXIMIZATION-STEP(φ,γ);
```

The $\gamma(z)$ variable is used to accumulate the expected number of times that the edit operation z was used to generate the string pairs in C. Convergence is achieved when the parameter values do not change on consecutive iterations. In practice, we typically terminate the algorithm when the increase in the total probability of the training corpus C falls below a fixed threshold. Alternately, we might simply perform a fixed number of iterations.

Let us now consider the details of the algorithm, beginning with the expectation step. First we define our forward and backward variables. The forward variable $\alpha_{t,v}$ contains the probability $p(x^t, y^v|\phi)$ of generating the pair $\langle x^t, y^v \rangle$ of prefixes. These values are calculated by the FORWARD-EVALUATE() algorithm given in the preceding section.

The following BACKWARD-EVALUATE() algorithm calculates the backward values. The backward variable $\beta_{t,v}$ contains the probability $p(x_{t+1}^T, y_{v+1}^V|\phi, \langle t, v \rangle)$ of generating the suffix pair $\langle x_{t+1}^T, y_{v+1}^V \rangle$. Note that $\beta_{0,0}$ is equal to $\alpha_{T,V}$.

```
BACKWARD-EVALUATE(x^T, y^V, φ)
1.  β_{T,V} := 1;
2.  for t = T ... 0
3.      for v = V ... 0
4.          if (v < V ∨ t < T) [ β_{t,v} := 0; ]
5.          if (v < V) [ β_{t,v} += δ(ε, y_{v+1}) β_{t,v+1}; ]
6.          if (t < T) [ β_{t,v} += δ(x_{t+1}, ε) β_{t+1,v}; ]
7.          if (v < V ∧ t < T) [ β_{t,v} += δ(x_{t+1}, y_{v+1}) β_{t+1,v+1}; ]
8.  return(β);
```

Finally, we let $\gamma(c)$ be the expected number of times the edit operation c was used to generate the string pair $\langle x^T, y^V \rangle$. These values are calculated by the following EXPECTATION-STEP() algorithm, which assumes that the $\gamma(\cdot)$ accumulators have been properly initialized.

EXPECTATION-STEP(x^T,y^V,ϕ,γ)
1. α := FORWARD-EVALUATE(x^T,y^V,ϕ);
2. β := BACKWARD-EVALUATE(x^T,y^V,ϕ);
3. for $t = 0 \ldots T$
4. for $v = 0 \ldots V$
5. if $(t > 0)$ [$\gamma(x_t, \epsilon)$ += $\alpha_{t-1,v}\delta(x_t, \epsilon)\beta_{t,v}/p(x^T, y^V|\phi)$;]
6. if $(v > 0)$ [$\gamma(\epsilon, y_v)$ += $\alpha_{t,v-1}\delta(\epsilon, y_v)\beta_{t,v}/p(x^T, y^V|\phi)$;]
7. if $(t > 0 \wedge v > 0)$ [$\gamma(x_t, y_v)$ += $\alpha_{t-1,v-1}\delta(x_t, y_v)\beta_{t,v}/p(x^T, y^V|\phi)$;]

Line 5 accumulates the posterior probability that we were in state $\langle t-1, v\rangle$ and emitted a $\langle x_t, \epsilon\rangle$ deletion operation. Similarly, line 6 accumulates the posterior probability that we were in state $\langle t, v-1\rangle$ and emitted a $\langle \epsilon, y_v\rangle$ insertion operation. Line 7 accumulates the posterior probability that we were in state $\langle t-1, v-1\rangle$ and emitted a $\langle x_t, y_v\rangle$ substitution operation.

Given the expectations of our edit operations, the following MAXIMIZATION-STEP() algorithm updates our model parameters.

MAXIMIZATION-STEP(ϕ,γ)
1. $N := 0$;
2. forall z in E [N += $\gamma(z)$;]
3. forall z in E [$\delta(z) := \gamma(z)/N$;]

The EXPECTATION-STEP() algorithm accumulates the expectations of edit operations by considering all possible generation sequences. It is possible to replace this algorithm with the VITERBI-EXPECTATION-STEP() algorithm, which accumulates the expectations of edit operations by only considering the single most likely generation sequence for a given pair of strings. The only change to the EXPECTATION-STEP() algorithm would be to replace the subroutine calls in lines 1 and 2. Although such a learning algorithm is arguably more appropriate to the original string edit distance formulation, it is less suitable in our stochastic model of string edit distance and so we do not pursue it here.

2.5 Three Variants

Here we briefly consider three variants of the memoryless stochastic transducer. First, we explain how to reduce the number of free parameters in the transducer, and thereby simplify the corresponding edit cost function. Next, we propose a way to combine different transduction distances using the technique of finite mixture modeling. Finally, we suggest an even stronger class of string distance functions that are based on stochastic transducers with memory.

2.5.1 Parameter Tying

In many applications, the edit cost function is simpler than the one that we have been considering here. For example, the most widely used edit distance

has only four distinct costs: the insertion cost, the deletion cost, the identity cost, and the substitution cost. Although this simplification may result in a weaker edit distance, it has the advantage of requiring less training data to accurately learn the edit costs. In the statistical modeling literature, the use of parameter equivalence classes is dubbed parameter tying.

It is straightforward to implement parameter tying for memoryless stochastic transducers. Let $\tau(z)$ be the equivalence class of the edit operation z, $\tau(z) \in 2^E$, and let $\delta(\tau(z)) = \sum_{z' \in \tau(z)} \delta(z')$ be the total probability assigned to the equivalence class $\tau(z)$. After estimation, we simply set $\delta(z)$ to be uniform within the total probability $\delta(\tau(z))$ assigned to $\tau(z)$.

$$\delta(z) := \delta(\tau(z))/|\tau(z)|$$

2.5.2 Finite Mixtures

A k-component mixture transducer $\phi = \langle A, B, \mu, \delta \rangle$ is a linear combination of k memoryless transducers defined on the same alphabets A and B. The mixing parameters μ form a probability function, where μ_i is the probability of choosing the i^{th} memoryless transducer. Therefore, the total probability assigned to a pair of string by a mixture transducer is a weighted sum over all the component transducers.

$$p(x^T, y^V | \phi) = \sum_{i=1}^{k} p(x^T, y^V | \langle A, B, \delta_i \rangle) \mu_i$$

A mixture transducer combines the predictions of its component transducers in a surprisingly effective way. Since the cost $-\log \mu_i$ of selecting the i^{th} component of a mixture transducer is insignificant when compared to the total cost $-\log p(x^T, y^V | \phi_i)$ of the string pair according to the i^{th} component, the string distance defined by a mixture transducer is effectively the minimum over the k distances defined by its k component transducers.

Choosing the components of a mixture transducer is more of an art than a science. One effective approach is to combine simpler models with more complex models. In this setting, this means that we would combine transducers with varying degrees of parameter tying, all trained on the same corpus. The mixing parameters could be uniform, ie., $\mu_i = 1/k$, or they could be optimized using withheld training data [8].

Another effective approach is to combine models trained on different corpora. This makes the most sense if the training corpus consists of naturally distinct sections. In this setting, we would train a different transducer on each section of the corpus, and then combine the resulting transducers into a mixture model. The mixing parameters could be set to the relative sizes of the corpus sections, or they could be optimized using withheld training data. (For good measure, we could also include a transducer that was trained on the entire corpus.)

2.5.3 Memory

From a statistical perspective, the memoryless transducer is quite weak because it models consecutive edit operations as being independent. A more powerful model – the stochastic transducer with memory – would condition the probability $\delta(z|\langle x_{t-n}^t, y_{v-n}^v\rangle)$ of generating an edit operation z on a finite suffix of the partial string pair $\langle x^t, y^v\rangle$ that has already been generated. Such a stochastic transducer can be further strengthened with state-conditional interpolation [8]. We could also condition our probabilities $\delta(z|\langle x_{t-n}^t, y_{v-n}^v\rangle, s)$ on a hidden state s drawn from a finite state space. The details of this approach are presented in forthcoming work.

3 An Application

In this section, we apply our techniques to the problem of learning the pronunciations of words. A given word of a natural language may be pronounced in many different ways, depending on such factors as the dialect, the speaker, and the linguistic environment. We describe one way of modeling variation in the pronunciation of words. Let W be the set of syntactic words in a language, let A be the set of underlying phonological segments employed by the language, and let B be the set of observed phonemes. The pronouncing lexicon $L : W \rightarrow 2^{A^*}$ assigns a small set of underlying phonological forms to every syntactic word in the language. Each underlying form in A^* is then mapped to a surface form in B^* by a stochastic process. Our goal is to recognize phonetic strings, which will require us to map each surface form to the syntactic word for which it is a pronunciation.

Let us formalize this pronunciation recognition (PR) problem as follows. The input to Pronunciation Recognition is a six-tuple $\langle W, A, B, L, C, C'\rangle$ consisting of a set W of syntactic words, an alphabet A of phonological segments, an alphabet B of phonetic segments, a pronouncing lexicon $L : W \rightarrow 2^{A^*}$, a training corpus $C = \langle w_1, y^{V_1}\rangle, \ldots, \langle w_n, y^{V_n}\rangle$ of labeled phonetic strings, and a testing corpus $C' = y^{S_1}, \ldots, y^{S_m}$ of unlabeled phonetic strings. Each training pair $\langle w_i, y^{V_i}\rangle$ in C includes a syntactic word w_i, $w_i \in W$, along with a phonetic string $y^{V_i} \in B^{V_i}$. The output is a set of labels $v_1, \ldots, v_m$ for the phonetic strings in the testing corpus C'.

Our solution to this pronunciation recognition problem is to explicitly model the joint probability of a lexical entry and its surface realizations. That is, we model the joint probability $p(w, x^t, y^v|\phi, L)$ of a lexical entry $\langle w, x^t\rangle \in L$ for a syntactic word w and a surface form y^v as a product of conditional probabilities

$$p(w, x^t, y^v|\phi, L) = p(y^v|x^t, \phi)p(w, x^t|\phi, L) \tag{5}$$

where the prior probability $p(w, x^t|\phi, L)$ of an entry $\langle w, x^t\rangle$ in the pronouncing lexicon L is estimated from the training corpus C, and the conditional probability $p(y^v|x^t, \phi) = p(y^v, x^t|\phi)/p(x^t|\phi)$ of a surface form y^v is determined by

a memoryless stochastic transducer ϕ. This is a giant finite mixture model where the $p(\langle w, x^t \rangle | \phi, L)$ are the mixing parameters and the $p(y^v | \langle x^t, w \rangle, \phi) = p(y^v | x^t, \phi)$ are the components of the mixture.

We estimate the parameters of our mixture model (5) using expectation maximization [4]. In brief, we accumulate the posterior probabilities $p(\langle w_i, x^t \rangle | y^{V_i}, w_i)$ for all $x^t \in L(w_i)$ that the lexical entry $\langle w_i, x^t \rangle$ generated the surface form y^{V_i} with label w_i. These expectations are used to reestimate the $p(\langle w, x^t \rangle | \phi, L)$ mixing parameters of our model. The accumulators for the mixing parameters were initialized to 0.1 so that no mixing parameter could be reestimated to zero. No attempt was made to optimize this flattening constant. We also weight our expectation accumulation for the component models $p(y^{V_i}, x^t | \phi)$ by this posterior probability. As a result, this training algorithm only trains the transduction distance on similar strings.

For each surface form y^v in the testing corpus C', we output $\hat{w}$

$$\begin{aligned}
\hat{w} &\doteq \mathsf{argmax}_w \{p(w|y^v, \phi, L)\} \\
&= \mathsf{argmax}_w \{p(w, y^v | \phi, L)/p(y^v | \phi, L)\} \\
&= \mathsf{argmax}_w \{p(w, y^v | \phi, L)\} \\
&= \mathsf{argmax}_w \left\{\sum_{x^t \in A^*} p(w, x^t, y^v | \phi, L)\right\} \\
&= \mathsf{argmax}_w \left\{\sum_{x^t \in L(w)} p(y^v | x^t, \phi) p(w, x^t | \phi, L)\right\}
\end{aligned}$$

where $L(w)$ is the set of permissible underlying forms for the word w. This decision rule correctly aggregates the similarity between a surface form and all lexical entries for a given word. Experimental results obtained using this stochastic approach are presented below.

In appendix A we consider a simpler approach to the pronunciation recognition problem based on the classic "nearest neighbor" decision rule. In this ad-hoc approach, we learn a string distance function using all valid pairs $\langle x^t, y^{V_i} \rangle$ of underlying forms $x^t \in L(w_i)$ and surface realizations y^{V_i} for each word w_i in the training corpus. For each phonetic string y^{S_j} in the testing corpus C', we return the word $\hat{v}_j$ in D that minimizes the string distance $d(x^t, y^{S_j})$ among all lexical entries $\langle v, x^t \rangle \in L$. Although this approach is technically simple, it has the unfortunate property of training the transduction distances on both similar and dissimilar pairs of strings. Consequently, the performance of the transduction distances trained using this approach are not appreciably different from the performance of the untrained Levenshtein distance. Experimental results obtained using this ad-hoc approach are presented in appendix A.

Let us now apply our stochastic approach to the Switchboard corpus of conversational speech.

3.1 Switchboard Corpus

The Switchboard corpus contains over 3 million words of spontaneous telephone speech conversations [5]. It is considered one of the most difficult corpora for

speech recognition (and pronunciation recognition) because of the tremendous variability of spontaneous speech. Current speech recognition technology has a word error rate above 45% on the Switchboard corpus. As a point of contrast, the same speech recognition technology achieves a word error rate of less than 5% on read speech.

Approximately 280,000 words of Switchboard have been manually assigned phonetic transcripts at ICSI using a proprietary phonetic alphabet [6]. The Switchboard corpus also includes a pronouncing lexicon with 71,100 entries using a modified Pronlex phonetic alphabet (long form) [1]. In order to make the pronouncing lexicon compatible with the ICSI corpus of phonetic transcripts, we removed 148 entries from the lexicon and 73,068 samples from the ICSI corpus.[3] After filtering, our pronouncing lexicon had 70,951 entries for 66,284 syntactic words over an alphabet of 42 phonemes. Our corpus had 214,310 samples – of which 23,955 were distinct – for 9,015 syntactic words with 43 phonemes (42 Pronlex phonemes plus a special "silence" symbol).

3.2 Four Experiments

We conducted four sets of experiments using seven models. In all cases, we partitioned our corpus of 214,310 samples 9:1 into 192,879 training samples and 21,431 test samples.

Our seven models consist of Levenshtein distance [10] as well as six variants resulting from our two interpretations of three models. Our two interpretations are the stochastic edit distance (4) and the classic edit distance (3), also called the Viterbi edit distance. For each interpretation, we built a tied model with only four parameters, an untied model, and a mixture model consisting of a uniform mixture of the tied and untied models. Model parameters were initialized uniformly before training.

Our four sets of experiments are determined by how we obtain our pronouncing lexicon. The first two experiments use the Switchboard pronouncing lexicon. Experiment E1 uses the full pronouncing lexicon for all 66,284 words while experiment E2 uses the subset of the pronouncing lexicon for the 9,015 words in the corpus. This subset has only 9,621 entries. The second two experiments use a lexicon constructed from the corpus. Experiment E3 uses the training corpus only to construct the pronouncing lexicon, while experiment E4 uses the entire corpus – both training and testing portions – to construct the pronouncing lexicon. The lexicon derived from the training corpus has 22,140 entries for 8,570 words. (The test corpus has 512 samples whose words did not appear in the

[3] From the lexicon, we removed 148 entries whose words had unusual punctuation ([<!.]). From the ICSI corpus, we removed 72,257 samples without a valid syntactic word, 688 samples with an empty phonetic transcript, 88 samples with a fragmentary transcript, 27 samples with the undocumented symbol ?, and 8 samples with the undocumented symbol !. Note that the symbols ? and ! are not part of either the ICSI phonetic alphabet or the Pronlex phonetic alphabet (long forms), and are only used in the ICSI corpus.

training corpus, which lower bounds the error rate on this experiment to 2.4%.) The lexicon derived from the entire corpus has 23,955 entries for 9,015 words.

The principal difference among these four experiments is how much information the training corpus provides about the test corpus. In order of increasing information, we have E3 < E1 < E2 < E4. In experiment E3, the pronouncing lexicon is constructed from the training corpus only and therefore the only information provided in E3 is the size of the phonetic alphabet. In experiment E1, the pronouncing lexicon was constructed from the entire 3m word Switchboard corpus, and therefore E1 also provides weak knowledge of the set of syntactic words that appear in the test corpus. In experiment E2, the pruned pronouncing lexicon provides strong knowledge of the set of syntactic words that actually appear in the test corpus, as well as their most salient phonetic forms. In experiment E4, the pronouncing lexicon provides complete knowledge of the set of syntactic words paired with their actual phonetic forms in the test corpus.

For each experiment, we report the fraction of misclassified samples in the testing corpus (ie., the word error rate). Note that the pronouncing lexicons have many homophones. Our decision rule $d : B^* \rightarrow 2^L$ maps each test sample y^{S_i} to a subset $d(y^{S_i}) \subset L$ of the lexical entries. Accordingly, we calculate the fraction of correctly classified samples as the sum over all test samples of the ratio of the number of correct lexical entries in $d(y^{S_i})$ to the total number of postulated lexical entries in $d(y^{S_i})$. The fraction of misclassified samples is one minus the fraction of correctly classified samples.

3.3 Results

Our experimental results for the stochastic approach described above are summarized in the following table and figures.

	Leven-	Stochastic Distance			Viterbi Distance		
	shtein	Tied	Untied	Mixed	Tied	Untied	Mixed
E1	48.04	18.96	15.47	16.11	18.88	**15.44**	16.11
E2	33.00	18.82	15.30	15.98	18.75	**15.26**	15.99
E3	61.87	24.01	**20.35**	21.77	23.98	20.36	21.76
E4	56.35	21.60	**15.98**	18.92	21.59	**15.98**	18.91

Table 1: Word error rate for seven string distance functions in four experiments. This table shows the best result for each model, as if an oracle told us when to stop training.

The initial performance of the transduction distances is extremely poor in all experiments because training starts with a uniform model, that assigns equal probability to all edit operations. After training, the error rates of the transduction distances are less than one third the error rate of the untrained Levenshtein

distance. The Viterbi edit distance performs slightly better than the stochastic edit distance in most but not all situations. The untied model performs significantly better than both the tied and mixed models in all four experiments.

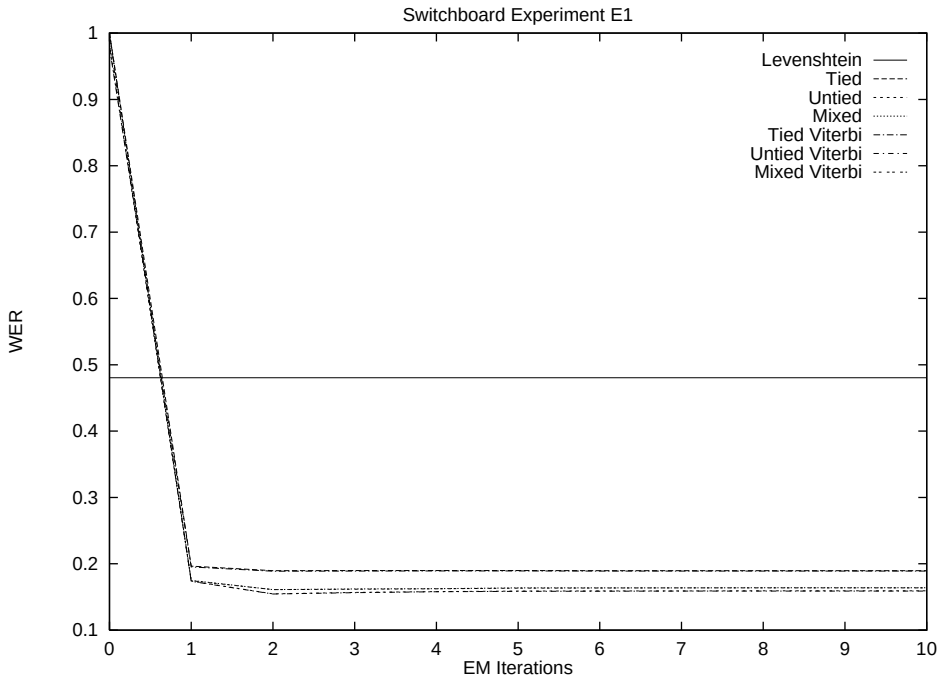

Figure 1: Word error rate as a function of EM iterations for seven models using the full Switchboard pronouncing lexicon (70,952 entries for 66,284 words and 52,513 forms). After only one EM iteration, all transduction distances have less than half the error rate of the Levenshtein distance. After two EM iterations, the untied and mixed models have less than one third the error rate of the Levenshtein distance.

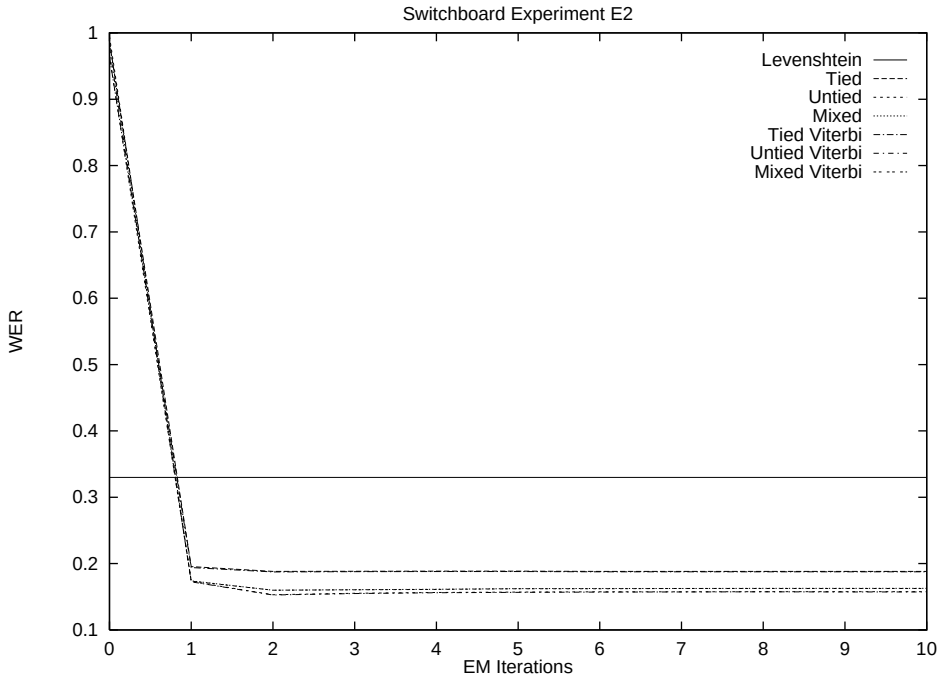

Figure 2: Word error rate as a function of EM iterations for seven models using the subset of the Switchboard pronouncing lexicon whose words are manifest in the ICSI corpus (9,621 entries for 9,015 words and 9,343 forms). The untied and mixed models have less than half the error rate of the Levenshtein distance.

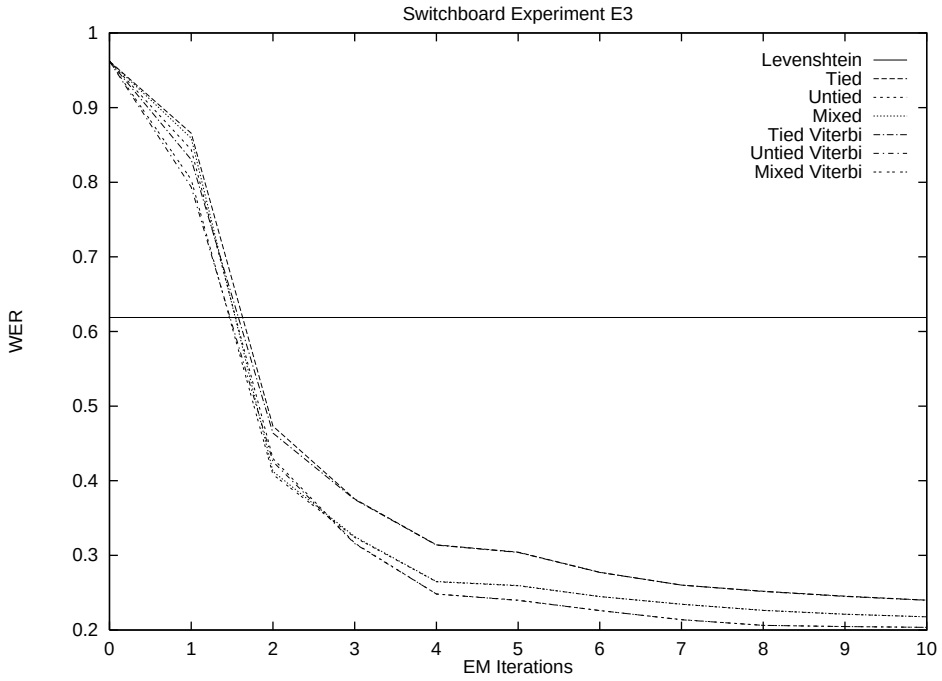

Figure 3: Word error rate as a function of EM iterations for seven models using a pronouncing lexicon derived from the ICSI training corpus (22,140 entries for 8,570 words and 17,880 forms). The test corpus has 512 samples whose words do not appear in the resulting lexicon. The untied models have less than one third the error rate of the Levenshtein distance. The three lines that emerge during later EM iterations correspond to the untied, mixed, and tied models, respectively. The performance of all six transduction distances is continuing to improve at 10 EM iterations.

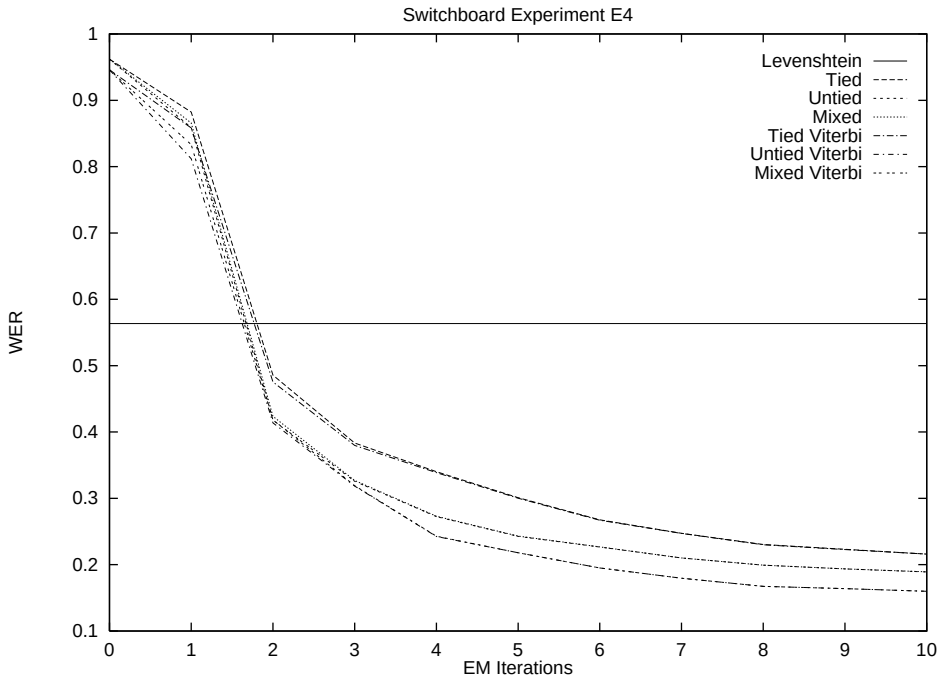

Figure 4: Word error rate as a function of EM iterations for seven models using a pronouncing lexicon derived from the entire ICSI corpus, including both training and testing portions (23,955 entries for 9,015 words and 19,355 forms. The untied and mixed models have less than one third the error rate of the Levenshtein distance. The three lines that emerge during later EM iterations correspond to the untied, mixed, and tied models, respectively. The performance of all six transduction distances is continuing to improve at 10 EM iterations.

References

[1] COMLEX pronouncing lexicon, version 0.2. Linguistic Data Consortium LDC95L3, July 1995.

[2] Baum, L., and Eagon, J. An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to models for ecology. *Bull. AMS 73* (1967), 360–363.

[3] Baum, L., Petrie, T., Soules, G., and Weiss, N. A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Stat. 41* (1970), 164–171.

[4] Dempster, A., Laird, N., and Rubin, D. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statist. Soc. Ser. B (methodological) 39* (1977), 1–38.

[5] Godfrey, J., Holliman, E., and McDaniel, J. Switchboard: telephone speech corpus for research and development. In *Proc. IEEE ICASSP* (Detroit, 1995), pp. 517–520.

[6] Greenberg, S., Hollenbach, J., and Ellis, D. Insights into spoken language gleaned from phonetic transcription of the Switchboard corpus. In *Proc. ICSLP* (Philadelphia, October 1996).

[7] Hall, P., and Dowling, G. Approximate string matching. *Computing Surveys 12*, 4 (1980), 381–402.

[8] Jelinek, F., and Mercer, R. L. Interpolated estimation of Markov source parameters from sparse data. In *Pattern Recognition in Practice* (Amsterdam, May 21–23 1980), E. S. Gelsema and L. N. Kanal, Eds., North Holland, pp. 381–397.

[9] Kukich, K. Techniques for automatically correcting words in text. *ACM Compute. Surveys 24* (1992), 377–439.

[10] Levenshtein, V. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics – Doklady 10*, 10 (1966), 707–710.

[11] Marzal, A., and Vidal, E. Computation of normalized edit distance and applications. *IEEE Trans. PAMI 15*, 9 (1993), 926–932.

[12] Masek, W., and Paterson, M. A faster algorithm computing string edit distances. *J. Comput. System Sci. 20* (1980), 18–31.

[13] Oomman, B. Constrained string editing. *Information Sciences 40* (1986), 267–284.

[14] Peterson, J. Computer programs for detecting and correcting spelling errors. *Comm. ACM 23* (1980), 676–687.

[15] Redner, R. A., and Walker, H. F. Mixture densities, maximum likelihood, and the EM algorithm. *SIAM Review 26*, 2 (1984), 195–239.

[16] Ristad, E. S., and Yianilos, P. N. Finite growth models. Tech. Rep. 533-96, Department of Computer Science, Princeton University, Princeton, NJ, October 1996.

[17] Sankoff, D., and Kruskal, J. B., Eds. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison.* Addison–Wesley, Reading, MA, 1983.

[18] Wagner, R., and Fisher, M. The string to string correction problem. *JACM 21* (1974), 168–173.

A An Ad-Hoc Solution

In this appendix we report experimental results for a simple but ad-hoc solution to the pronunciation recognition problem based on the classic "nearest neighbor" decision rule. Here we learn a string distance function using all valid pairs $\langle x^t, y^{V_i} \rangle$ of underlying forms $x^t \in L(w_i)$ and surface realizations y^{V_i} for each word w_i in the training corpus. For each phonetic string y^{S_j} in the testing corpus C', we return the word $\hat{v}_j$ in D that minimizes the string distance $d(x^t, y^{S_j})$ among all lexical entries $\langle v, x^t \rangle \in L$.

Our results are presented in the following table and figures. The most striking property of these results is how poorly the trained transduction distances perform relative to the simple Levenshtein distance, particularly when the pronouncing lexicon is derived from the corpus (experiments E3 and E4).

	Leven-	Stochastic Distance			Viterbi Distance		
	shtein	Tied	Untied	Mixed	Tied	Untied	Mixed
E1	48.04	48.39	46.78	46.95	48.41	**46.75**	46.91
E2	33.00	33.51	**31.54**	31.82	33.68	31.55	31.81
E3	61.87	62.80	61.89	62.31	62.89	**61.86**	62.26
E4	**56.35**	**56.35**	56.73	56.36	**56.35**	56.73	**56.35**

Table 2: Word error rate for seven string distance functions in four experiments. This table shows the best result for each model, as if an oracle told us when to stop training. None of the transduction distances is significantly better than the untrained Levenshtein distance in this approach.

We believe that the poor performance of our transduction distances in these experiments is due to the simple ad-hoc training paradigm. The handcrafted lexicon used in experiments E1 and E2 contains only 1.06 entries per syntactic word. In contrast, the lexicon derived from the corpus contains 2.66 entries per syntactic word. These entries can be quite dissimilar, and so our ad-hoc training paradigm trains our transduction distances on both similar and dissimilar strings. The results presented in section 3.3 confirm this hypothesis.

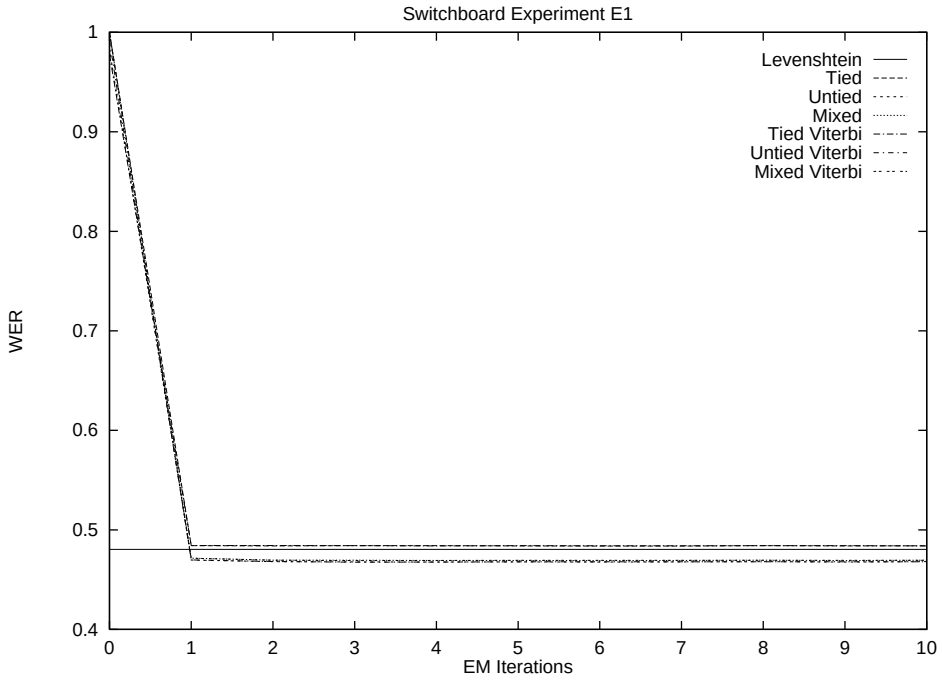

Figure 5: Word error rate as a function of EM iterations for seven models using the full Switchboard pronouncing lexicon (70,952 entries for 66,284 words and 52,513 forms). Both untied and mixed models do slightly better than the untrained Levenshtein distance, while the tied models do slightly worse.

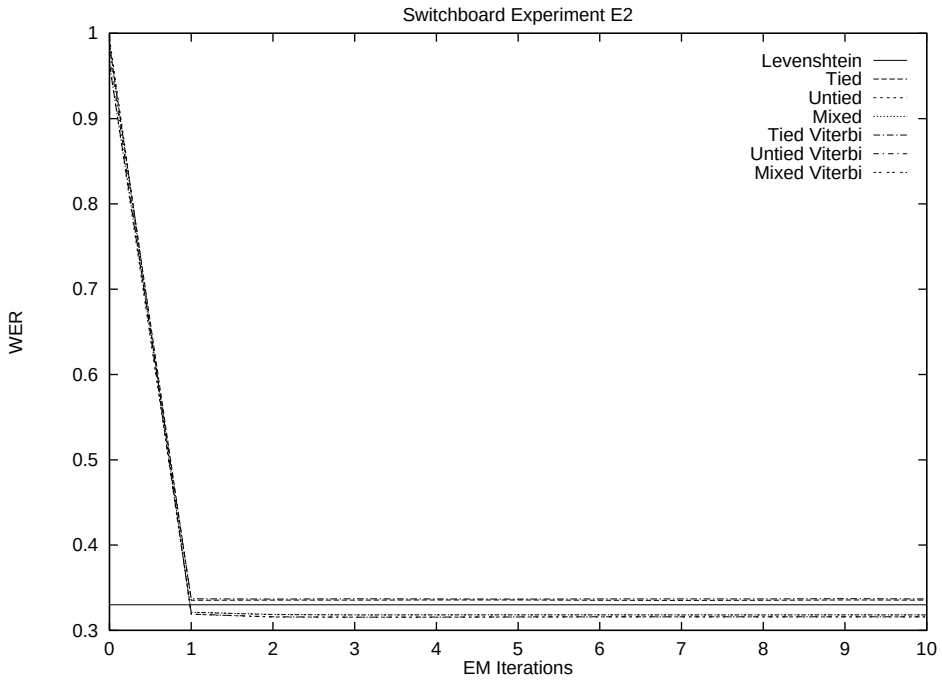

Figure 6: Word error rate as a function of EM iterations for seven models using the subset of the Switchboard pronouncing lexicon whose words are manifest in the ICSI corpus (9,621 entries for 9,015 words and 9,343 forms). Both untied and mixed models do slightly better than the untrained Levenshtein distance, while the tied models do slightly worse.

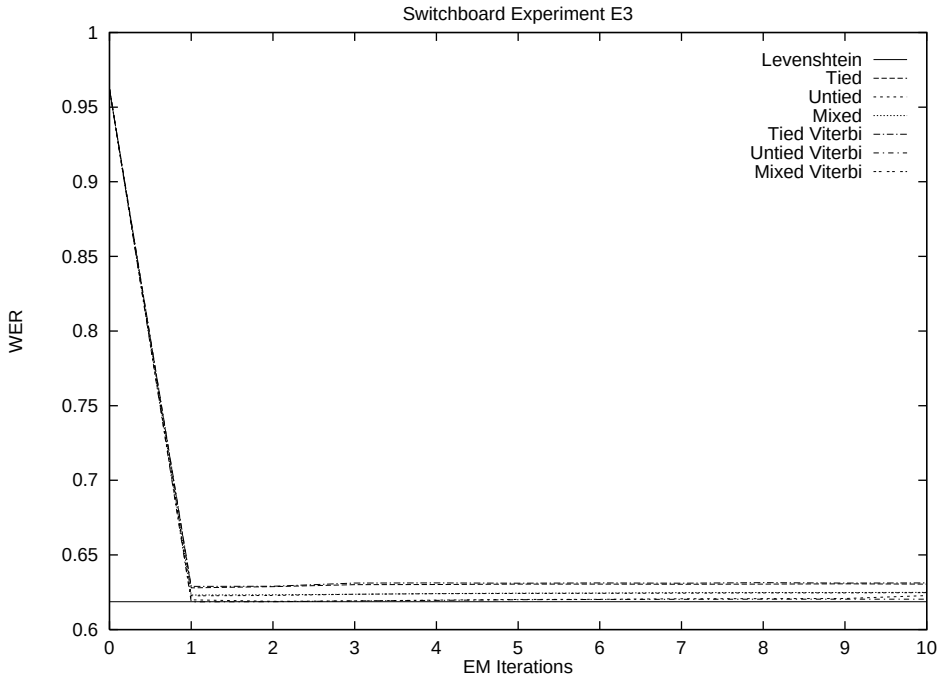

Figure 7: Word error rate as a function of EM iterations for seven models using a pronouncing lexicon derived from the ICSI training corpus (22,140 entries for 8,570 words and 17,880 forms). The test corpus has 512 samples whose words do not appear in the resulting lexicon. No model does better than the untrained Levenshtein distance.

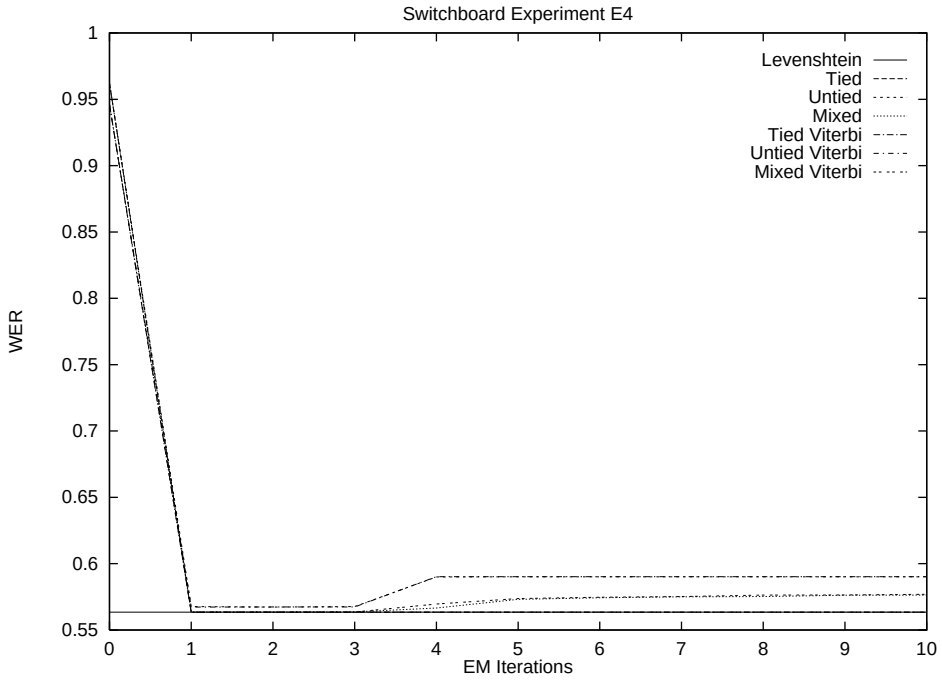

Figure 8: Word error rate as a function of EM iterations for seven models using a pronouncing lexicon derived from the entire ICSI corpus, including both training and testing portions (23,955 entries for 9,015 words and 19,355 forms. No model does better than the untrained Levenshtein distance, although the tied models equal its performance. Additional training significantly decreases the performance of the untied and mixed models.

B Conditioning on String Lengths

In the main body of this report, we presented an approach to memoryless stochastic transduction that was based on maximal prefix-free sets. In this appendix, we define a probability function $p(\cdot, \cdot|\theta, T, V)$ over all strings of lengths T and V. Thus, unlike probability function $p(\cdot, \cdot|\phi)$ defined in the main body of this report, summing $p(\cdot, \cdot|\theta, T, V)$ over all pairs of strings in $A^T \times B^V$ will result in unity.

$$\sum_{x^T \in A^T, y^V \in B^V} p(x^T, y^V|\theta, T, V) = 1$$

As we shall see, the approach pursued in the body of the report has the advantage of a simpler parameterization and simpler algorithms. A second difference between the two approaches is that the former approach learns the relative lengths of the string pairs in the training corpus while the latter approach cannot. Thus, the approach developed in this appendix is appropriate when we wish to ignore the relative lengths of the string pairs in the training corpus.

We begin by presenting an alternate parameterization of the memoryless transducer, where the edit probabilities δ are factored into the product of a marginal probability ω and a conditional probability δ. Next, we present algorithms that generate, evaluate, and learn the parameters for finite strings, conditioned on their lengths.

B.1 Parameterization

A *factored memoryless transducer* $\theta = \langle A, B, \omega, \delta \rangle$ consists of two finite alphabets A and B as well as the triple $\omega = \langle \omega_d, \omega_i, \omega_s \rangle$ of transition probabilities and the triple $\delta = \langle \delta_d, \delta_i, \delta_s \rangle$ of observation probabilities. ω_s is the probability of generating a substitution operation and $\delta_s(a, b)$ is the probability of choosing the particular symbols a and b to substitute. Similarly, ω_d is the probability of generating a deletion operation and $\delta_d(a)$ is the probability of choosing the symbol a to delete, while ω_i is the probability of generating a insertion operation and $\delta_i(b)$ is the probability of choosing the symbol b to insert.

The translation from our factored parameterization $\langle A, B, \omega, \delta \rangle$ back to our unfactored parameterization $\langle A, B, \delta \rangle$ is straightforward.

$$\begin{aligned} \delta(a, \epsilon) &= \omega_d \delta_d(a) \\ \delta(\epsilon, b) &= \omega_i \delta_i(b) \\ \delta(a, b) &= \omega_s \delta_s(a, b) \end{aligned}$$

The translation from the unfactored parameterization to the factored pa-

rameterization is also straightforward.

$$\begin{array}{lcl} \omega_d & = & \sum_{e \in E_d} \delta(e) \\ \delta_d(a) & = & \delta(a, \epsilon)/\omega_d \\ \\ \omega_i & = & \sum_{e \in E_i} \delta(e) \\ \delta_i(b) & = & \delta(\epsilon, b)/\omega_i \\ \\ \omega_s & = & \sum_{e \in E_s} \delta(e) \\ \delta_s(a,b) & = & \delta(a,b)/\omega_s \end{array}$$

As explained below, the factored parameterization is required in order to properly accumulate expectations when the expectation maximization algorithm is conditioned on the string lengths.

B.2 Generation

A factored memoryless transducer $\theta = \langle A, B, \omega, \delta \rangle$ induces a probability function $p(\cdot, \cdot | \theta, T, V)$ on the joint space $A^T \times B^V$ of all pairs of strings of length T and V. This probability function is defined by the following algorithm, which generates a string pair $\langle x^T, y^V \rangle$ from the joint space $A^T \times B^V$ according to $p(\cdot | \theta, T, V)$.

GENERATE-STRINGS(T,V,θ)
1. initialize $t := 1$; $v := 1$;
2. while $t \leq T$ and $v \leq V$
3. pick $\langle a, b \rangle$ from E according to $\delta(\cdot)$
4. if $(a \in A)$ then $x_t := a$; $t := t + 1$;
5. if $(b \in B)$ then $y_v := b$; $v := v + 1$;
6. while $t \leq T$
7. pick a from A according to $\delta_d(\cdot)$
8. $x_t := a$; $t := t + 1$;
9. while $v \leq V$
10. pick b from B according to $\delta_i(\cdot)$
11. $y_v := b$; $v := v + 1$;
12. return($\langle x^T, y^V \rangle$);

The GENERATE-STRINGS() algorithm begins by drawing edit operations from E according to the edit probability $\delta(\cdot)$ until at least one of the partial strings x^t and y^v is complete [lines 2-5]. If y^v is complete but x^t is incomplete, then we complete x^t using symbols drawn from A according to the marginal probability $\delta_d(\cdot) = \delta(\cdot | E_d)$ [lines 6-8]. Conversely, if x^t is complete but y^v is incomplete, then we complete y^v using symbols drawn from B according to the marginal $\delta_i(\cdot) = \delta(\cdot | E_i)$ [lines 9-11].

B.3 Evaluation

The marginal probability $p(x^T, y^V | \theta, T, V)$ of a pair of strings is calculated by summing the joint probability $p(x^T, y^V, z^n | \theta, T, V)$ over all the edit sequences that could have generated those strings.

$$\begin{aligned} p(x^T, y^V | \theta, T, V) &= \sum_{z^n} p(x^T, y^V, z^n | \theta, T, V) \\ &= \sum_{\{z^n : \rho(z^n) = \{x^T, y^V\}\}} p(z^n | \phi, T, V) \end{aligned}$$

where

$$\begin{aligned} p(z^n | \theta, T, V) &= \prod_i p(z_i | \theta, T, V, \rho(z^{i-1})) \\ &= \prod_i p(z_i | \theta, T, V, |\rho(z^{i-1})|) \end{aligned}$$

because the memoryless stochastic transducer is conditioned only on the string lengths T, V, and

$$p(z_i | \theta, T, V, t, v) = \begin{cases} 0 & \text{if } v = V \wedge t = T \\ \delta_d(a) & \text{if } v = V \wedge z_i = \langle a, \epsilon \rangle \\ \delta_i(b) & \text{if } t = T \wedge z_i = \langle \epsilon, b \rangle \\ \omega_s \delta_s(a, b) & \text{if } t < T \wedge v < V \wedge z_i = \langle a, b \rangle \\ \omega_d \delta_d(a) & \text{if } t < T \wedge v < V \wedge z_i = \langle a, \epsilon \rangle \\ \omega_i \delta_i(b) & \text{if } t < T \wedge v < V \wedge z_i = \langle \epsilon, b \rangle \end{cases}$$

by the definition of the GENERATE-STRINGS() algorithm.

Note that the corresponding distance functions

$$\begin{aligned} d_\theta(x^T, y^V | T, V) &\doteq -\log \mathsf{argmax}_{\{z^n : \rho(z^n) = \langle x^T, y^V \rangle\}} \{p(z^n | \phi, T, V)\} \\ d'_\theta(x^T, y^V | T, V) &\doteq -\log p(x^T, y^V | \theta, T, V) \end{aligned}$$

are now conditioned on the string lengths, and therefore are finite-valued only for strings in $A^T \times B^V$.

The following algorithms calculate the probability $p(x^T, y^V | \theta, T, V)$ in quadratic time and space $O(T \cdot V)$. The only difference between these versions and their unconditional variants in the body of the report is that conditioning on the string lengths T, V requires us to use the conditional probabilities $\delta_d(\cdot)$ and $\delta_i(\cdot)$ instead of the edit probabilities $\delta(\cdot)$ when a given hidden state sequence has completely generated one of the strings.

The following algorithm calculates the forward values. The forward variable $\alpha_{t,v}$ contains the probability $p(x^t, y^v | \theta, T, V)$ of generating the prefixes x^t and y^v of the finite string pair $\langle x^T, y^V \rangle$.

FORWARD-EVALUATE-STRINGS(x^T, y^V, θ)

1. $\alpha_{0,0} := 1;$
2. for $t = 1 \ldots T$ [$\alpha_{t,0} := \omega_d \delta_d(x_t) \alpha_{t-1,0};$]
3. for $v = 1 \ldots V$ [$\alpha_{0,v} := \omega_i \delta_i(y_v) \alpha_{0,v-1};$]
4. for $t = 1 \ldots T-1$
5. For $v = 1 \ldots V-1$
6. $\alpha_{t,v} := \omega_s \delta_s(x_t, y_v) \alpha_{t-1,v-1} + \omega_d \delta_d(x_t) \alpha_{t-1,v} + \omega_i \delta_i(y_v) \alpha_{t,v-1};$
7. for $t = 1 \ldots T-1$
8. $\alpha_{t,V} := \omega_s \delta_s(x_t, y_V) \alpha_{t-1,V-1} + \delta_d(x_t) \alpha_{t-1,V} + \omega_i \delta_i(y_V) \alpha_{t,V-1};$
9. for $v = 1 \ldots V-1$
10. $\alpha_{T,v} := \omega_s \delta_s(x_T, y_v) \alpha_{T-1,v-1} + \omega_d \delta_d(x_T) \alpha_{T-1,v} + \delta_i(y_v) \alpha_{T,v-1};$
11. $\alpha_{T,V} := \omega_s \delta_s(x_T, y_V) \alpha_{T-1,V-1} + \delta_d(x_T) \alpha_{T-1,V} + \delta_i(y_v) \alpha_{T,V-1};$
12. return(α);

The following algorithm calculates the backward values. The backward variable $\beta_{t,v}$ contains the probability $p(x_{t+1}^T, y_{v+1}^V | \theta, T, V, \langle t, v \rangle)$ of generating the suffixes $\langle x_{t+1}^T, y_{v+1}^V \rangle$ from the state $\langle t, v \rangle$.

BACKWARD-EVALUATE-STRINGS(x^T, y^V, θ)

1. $\beta_{T,V} := 1;$
2. for $t = T-1 \ldots 0$ [$\beta_{t,V} := \delta_d(x_{t+1}) \beta_{t+1,V};$]
3. for $v = V-1 \ldots 0$ [$\beta_{T,v} := \delta_i(y_{v+1}) \beta_{T,v+1};$]
4. for $t = T-1 \ldots 0$
5. for $v = V-1 \ldots 0$
6. $\beta_{t,v} := \omega_s \delta_s(x_{t+1}, y_{v+1}) \beta_{t+1,v+1} + \omega_d \delta_d(x_{t+1}) \beta_{t+1,v} + \omega_i \delta_i(y_{v+1}) \beta_{t,v+1};$
7. return(β);

B.4 Estimation

The principal difference between the two expectation step algorithms is that EXPECTATION-STEP-STRINGS() must accumulate expectations for the ω and δ parameter sets separately. Note that we may only accumulate expectations for the ω transition parameters when no transitions are forced.

EXPECTATION-STEP-STRINGS(x^T,y^V,θ,χ,γ)

1. α := FORWARD-EVALUATE(x^T,y^V,θ);
2. β := BACKWARD-EVALUATE(x^T,y^V,θ);
3. for $t = 1 \ldots T-1$
4. for $v = 1 \ldots V-1$
5. $m_s := \alpha_{t-1,v-1}\omega_s\delta_s(x_t, y_v)\beta_{t,v}/p(x^T, y^V|\theta, T, V)$;
6. $\gamma_s(x_t, y_v)$ += m_s; χ_s += m_s;
7. $m_d := \alpha_{t-1,v}\omega_d\delta_d(x_t)\beta_{t,v}/p(x^T, y^V|\theta, T, V)$;
8. $\gamma_d(x_t)$ += m_d; χ_d += m_d;
9. $m_i := \alpha_{t,v-1}\omega_i\delta_i(y_v)\beta_{t,v}/p(x^T, y^V|\theta, T, V)$;
10. $\gamma_i(y_v)$ += m_i; χ_i += m_i;
11. for $t = 1 \ldots T-1$ [$\gamma_d(x_t)$ += $\alpha_{t-1,V}\delta_d(x_t)\beta_{t,V}/p(x^T, y^V|\theta, T, V)$;]
12. for $v = 1 \ldots V-1$ [$\gamma_i(y_v)$ += $\alpha_{T,v-1}\delta_i(y_v)\beta_{T,v}/p(x^T, y^V|\theta, T, V)$;]

Lines 5-6 accumulate the posterior probability that we were in state $\langle t-1, v-1\rangle$ and emitted a $\langle x_t, y_v\rangle$ substitution operation. Lines 7-8 accumulate the posteriori probability that we were in state $\langle t-1, v\rangle$ and emitted a $\langle x_t, \epsilon\rangle$ deletion operation. Similarly, lines 9-10 accumulate the posteriori probability that we were in state $\langle t, v-1\rangle$ and emitted a $\langle \epsilon, y_v\rangle$ insertion operation. Lines 11 and 12 accumulate the corresponding posteriori probabilities for forced transitions.

Given the expectations of our transition parameters and observation parameters, the following MAXIMIZATION-STEP-STRINGS() algorithm updates our model parameters.

MAXIMIZATION-STEP-STRINGS(θ,χ,γ)

1. $N := \chi_d + \chi_i + \chi_s$;
2. $\omega_d := \chi_d/N$; $\omega_i := \chi_i/N$; $\omega_s := \chi_s/N$;
3. $N_d := 0$; forall a in A [N_d+ = $\gamma_d(a)$;]
4. forall a in A [$\delta_d(a) := \gamma_d(a)/N_d$;]
5. $N_i := 0$; forall b in B [N_i+ = $\gamma_i(b)$;]
6. forall b in B [$\delta_i(b) := \gamma_i(b)/N_i$;]
7. $N_s := 0$; forall $\langle a, b\rangle$ in $A \times B$ [N_s+ = $\gamma_s(a, b)$;]
8. forall $\langle a, b\rangle$ in $A \times B$ [$\delta_s(a, b) := \gamma_s(a, b)/N_s$;]