A Dynamic Bayesian Framework to Model Context and Memory in Edit Distance Learning: An Application to Pronunciation Classification

Karim Filali and **Jeff Bilmes***
Departments of Computer Science & Engineering and Electrical Engineering
University of Washington
Seattle, WA 98195, USA
{karim@cs,bilmes@ee}.washington.edu

Abstract

Sitting at the intersection between statistics and machine learning, Dynamic Bayesian Networks have been applied with much success in many domains, such as speech recognition, vision, and computational biology. While Natural Language Processing increasingly relies on statistical methods, we think they have yet to use Graphical Models to their full potential. In this paper, we report on experiments in learning edit distance costs using Dynamic Bayesian Networks and present results on a pronunciation classification task. By exploiting the ability within the DBN framework to rapidly explore a large model space, we obtain a 40% reduction in error rate compared to a previous transducer-based method of learning edit distance.

1 Introduction

Edit distance (ED) is a common measure of the similarity between two strings. It has a wide range of applications in classification, natural language processing, computational biology, and many other fields. It has been extended in various ways; for example, to handle simple (Lowrance and Wagner, 1975) or (constrained) block transpositions (Leusch et al., 2003), and other types of block operations (Shapira and Storer, 2003); and to measure similarity between graphs (Myers et al., 2000; Klein, 1998) or automata (Mohri, 2002).

*This material was supported by NSF under Grant No. ISS-0326276.

Another important development has been the use of data-driven methods for the automatic learning of edit costs, such as in (Ristad and Yianilos, 1998) in the case of string edit distance and in (Neuhaus and Bunke, 2004) for graph edit distance.

In this paper we revisit the problem of learning string edit distance costs within the Graphical Models framework. We apply our method to a pronunciation classification task and show significant improvements over the standard Levenshtein distance (Levenshtein, 1966) and a previous transducer-based learning algorithm.

In section 2, we review a stochastic extension of the classic string edit distance. We present our DBN-based edit distance models in section 3 and show results on a pronunciation classification task in section 4. In section 5, we discuss the computational aspects of using our models. We end with our conclusions and future work in section 6.

2 Stochastic Models of Edit Distance

Let $s_1^m = s_1 s_2 ... s_m$ be a *source* string over a source alphabet $\mathcal{A}$, and m the length of the string. s_i^j is the substring $s_i ... s_j$ and s_i^j is equal to the empty string, ϵ, when $i > j$. Likewise, t_1^n denotes a *target* string over a target alphabet $\mathcal{B}$, and n the length of t_1^n.

A source string can be transformed into a target string through a sequence of *edit operations*. We write $\langle s, t \rangle$ ($(s,t) \neq (\epsilon, \epsilon)$) to denote an *edit operation* in which the symbol s is replaced by t. If $s = \epsilon$ and $t \neq \epsilon$, $\langle s, t \rangle$ is an *insertion*. If $s \neq \epsilon$ and $t = \epsilon$, $\langle s, t \rangle$ is a *deletion*. When $s \neq \epsilon$, $t \neq \epsilon$ and $s \neq t$, $\langle s, t \rangle$ is a *substitution*. In all other cases, $\langle s, t \rangle$ is an *identity*.

The string edit distance, $d(s_1^m, t_1^n)$ between s_1^m and t_1^n is defined as the minimum weighted sum of

Proceedings of the 43rd Annual Meeting of the ACL, pages 338–345,
Ann Arbor, June 2005.

the number of deletions, insertions, and substitutions required to transform s_1^m into t_1^n (Wagner and Fischer, 1974). A $O(m \cdot n)$ Dynamic Programming (DP) algorithm exists to compute the ED between two strings. The algorithm is based on the following recursion:

$$d(s_1^i, t_1^j) = min \begin{pmatrix} d(s_1^{i-1}, t_1^j) + \gamma(\langle s_i, \epsilon \rangle), \\ d(s_1^i, t_1^{j-1}) + \gamma(\langle \epsilon, t_j \rangle), \\ d(s_1^{i-1}, t_1^{j-1}) + \gamma(\langle s_i, t_j \rangle) \end{pmatrix}$$

with $d(\epsilon, \epsilon)=0$ and $\gamma : \{\langle s, t \rangle | (s,t) \neq (\epsilon, \epsilon)\} \rightarrow \Re_+$ a cost function. When γ maps non-identity edit operations to unity and identities to zero, string ED is often referred to as the *Levenshtein distance*.

To learn the edit distance costs from data, Ristad and Yianilos (1998) use a generative model (henceforth referred to as the *RY model*) based on a memoryless transducer of string pairs. Below we summarize their main idea and introduce our notation, which will be useful later on.

We are interested in modeling the joint probability $P(S_1^m{=}s_1^m, T_1^n{=}t_1^n \,|\, \theta)$ of observing the source/target string pair (s_1^m, t_1^n) given model parameters θ. S_i (resp. T_i), $1 \leq i \leq m$, is a random variable (RV) associated with the event of observing a source (resp. target) symbol at position i.[1]

To model the edit operations, we introduce a hidden RV, Z, that takes values in $(\mathcal{A} \cup \epsilon \times \mathcal{B} \cup \epsilon) \setminus \{(\epsilon, \epsilon)\}$. Z can be thought of as a *random vector* with two components, $Z^{(s)}$ and $Z^{(t)}$.

We can then write the joint probability $P(s_1^m, t_1^n \,|\, \theta)$ as

$$P(s_1^m, t_1^n \,|\, \theta) = \sum_{\{z_1^\ell : v(z_1^\ell) = <s_1^m, t_1^n>,\, max(m,n) \leq \ell \leq m+n\}} \sum P(Z_1^\ell {=} z_1^\ell, s_1^m, t_1^n \,|\, \theta) \quad (1)$$

where $v(z_1^\ell)$ is the *yield* of the sequence z_1^ℓ: the string pair output by the transducer.

Equation 1 says that the probability of a particular pair of strings is equal to the sum of the probabilities of all possible ways to generate the pair by concatenating the edit operations $z_1 ... z_\ell$. If we make the assumption that there is no dependence between edit operations, we call our model *memoryless*. $P(Z_1^\ell, s_1^m, t_1^n \,|\, \theta)$ can then be factored as $\Pi_i P(Z_i, s_1^m, t_1^n \,|\, \theta)$. In addition, we call the model *context-independent* if we can write $Q(z_i) = P(Z_i {=} z_i, s_1^m, t_1^n \,|\, \theta)$, $1 < i < \ell$, where $z_i = \langle z_i^{(s)}, z_i^{(t)} \rangle$, in the form

$$Q(z_i) \propto \begin{cases} f^{ins}(t_{b_i}) & \text{for } z_i^{(s)} = \epsilon; z_i^{(t)} = t_{b_i} \\ f^{del}(s_{a_i}) & \text{for } z_i^{(s)} = s_{a_i}; z_i^{(t)} = \epsilon \\ f^{sub}(s_{a_i}, t_{b_i}) & \text{for } (z_i^{(s)}, z_i^{(t)}) = (s_{a_i}, t_{b_i}) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $\sum_z Q(z) = 1$; $a_i = \sum_{j=1}^{i-1} 1_{\{z_j^{(s)} \neq \epsilon\}}$ (resp. b_i) is the index of the source (resp. target) string generated up to the ith edit operation; and f^{ins},f^{del},and f^{sub} are functions mapping to $[0, 1]$.[2] Context independence is not to be taken here to mean Z_i does not depend on s_{a_i} or t_{b_i}. It depends on them through the *global context* which forces Z_1^ℓ to generate (s_1^m, t_1^n). The RY model is *memoryless and context-independent* (MCI).

Equation 2, also implicitly enforces the *consistency constraint* that the pair of symbols output, $(z_i^{(s)}, z_i^{(t)})$, agrees with the actual pair of symbols, (s_{a_i}, t_{b_i}), that needs to be generated at step i in order for the total yield, $v(z_1^\ell)$, to equal the string pair.

The RY stochastic model is similar to the one introduced earlier by Bahl and Jelinek (1975). The difference is that the Bahl model is memoryless and *context-dependent* (MCD); the f functions are now indexed by s_{a_i} (or t_{a_i}, or both) such that $\sum_z Q_{s_{a_i}}(z) = 1 \; \forall s_{a_i}$. In general, context dependence can be extended to include up to the whole source (and/or target) string, $s_1^{a_i - 1}, s_{a_i}, s_{a_i+1}^m$. Several other types of dependence can be exploited as will be discussed in section 3.

Both the Ristad and the Bahl transducer models give exponentially smaller probability to longer strings and edit sequences. Ristad presents an alternate explicit model of the joint probability of the length of the source and target strings. In this parametrization the probability of the length of an edit sequence does not necessarily decrease geometrically. A similar effect can be achieved by modeling the length of the hidden edit sequence explicitly (see section 3).

3 DBNs for Learning Edit Distance

Dynamic Bayesian Networks (DBNs), of which Hidden Markov Models (HMMs) are the most fa-

[1] We follow the convention of using capital letters for random variables and lowercase letters for instantiations of random variables.

[2] By convention, $s_{a_i} = \epsilon$ for $a_i > m$. Likewise, $t_{b_i} = \epsilon$ if $b_i > n$. $f^{ins}(\epsilon) = f^{del}(\epsilon) = f^{sub}(\epsilon, \epsilon) = 0$. This takes care of the case when we are past the end of a string.

mous representative, are well suited for modeling stochastic temporal processes such as speech and neural signals. DBNs belong to the larger family of Graphical Models (GMs). In this paper, we restrict ourselves to the class of DBNs and use the terms DBN and GM interchangeably. For an example in which Markov Random Fields are used to compute a context-sensitive edit distance see (Wei, 2004).[3]

There is a large body of literature on DBNs and algorithms associated with them. To briefly define a graphical model, it is a way of representing a (*factored*) probability distribution using a graph. Nodes of the graph correspond to random variables; and edges to dependence relations between the variables.[4] To do *inference* or parameter learning using DBNs, various generic exact or approximate algorithms exist (Lauritzen, 1996; Murphy, 2002; Bilmes and Bartels, 2003). In this section we start by introducing a graphical model for the MCI transducer then present four additional classes of DBN models: context-dependent, memory (where an edit operation can depend on past operations), direct (HMM-like), and length models (in which we explicitly model the length of the sequence of edits to avoid the exponential decrease in likelihood of longer sequences). A few other models are discussed in section 4.2.

3.1 Memoryless Context-independent Model

Fig. 1 shows a DBN representation of the memoryless context-independent transducer model (section 2). The graph represents a *template* which consists, in general, of three parts: a *prologue*, a *chunk*, and an *epilogue*. The chunk is repeated as many times as necessary to model sequences of arbitrary length. The product of *unrolling* the template is a Bayesian Network organized into a given number of *frames*. The prologue and the epilogue often differ from the chunk because they model boundary conditions, such as ensuring that the end of both strings is reached at or before the last frame.

Associated with each node is a probability function that maps the node's parent values to the values the node can take. We will refer to that function as a *conditional probability table* (CPT).

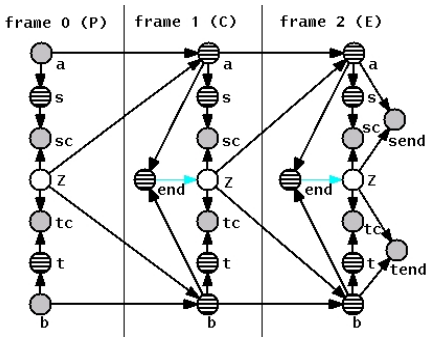

Figure 1: *DBN for the memory-less transducer model. Unshaded nodes are hidden nodes with probabilistic dependencies with respect to their parents. Nodes with stripes are deterministic hidden nodes, i.e., they take a unique value for each configuration of their parents. Filled nodes are observed (they can be either stochastic or deterministic). The graph template is divided into three frames. The center frame is repeated $m+n-2$ times to yield a graph with a total of $m+n$ frames, the maximum number of edit operations needed to transform s_1^m into t_1^n. Outgoing light edges mean the parent is a switching variable with respect to the child: depending on the value of the switching RV, the child uses different CPTs and/or a different parent set.*

Common to all the frames in fig. 1, are position RVs, a and b, which encode the current positions in the source and target strings resp.; source and target symbols, s and t; the hidden edit operation, Z; and consistency nodes sc and tc, which enforce the consistency constraint discussed in section 2. Because of symmetry we will explain the upper half of the graph involving the source string unless the target half is different. We drop subscripts when the frame number is clear from the context.

In the first frame, a and b are observed to have value 1, the first position in both strings. a and b determine the value of the symbols s and t. Z takes a random value $\langle z^{(s)}, z^{(t)} \rangle$. sc has the fixed observed value 1. The only configurations of its parents, Z and s, that satisfy $P(sc=1|s,z) > 0$ are such that ($Z^{(s)} = s$) or ($Z^{(s)} = \epsilon$ and $Z \neq \langle \epsilon, \epsilon \rangle$). This is the consistency constraint in equation 2.

In the following frame, the position RV a_2 depends on a_1 and Z_1. If Z_1 is an insertion (i.e. $Z_1^{(s)} = \epsilon$: the source symbol in the first frame is

[3] While the *Markov Edit Distance* introduced in the paper takes local statistical dependencies into account, the edit costs are still fixed and not corpus-driven.

[4] The concept of *d-separation* is useful to read independence relations encoded by the graph (Lauritzen, 1996).

not output), then a_2 retains the same value as a_1; otherwise a_2 is incremented by 1 to point to the next symbol in the source string.

The end RV is an indicator of when we are past the end of both source and target strings ($a > m$ and $b > n$). end is also a *switching parent* of Z; when $end = 0$, the CPT of Z is the same as described above: a distribution over edit operations. When $end = 1$, Z takes, with probability 1, a fixed value outside the range of edit operations but consistent with s and t. This ensures 1) no "null" state ($\langle \epsilon, \epsilon \rangle$) is required to fill in the value of Z until the end of the graph is reached; our likelihoods and model parameters therefore do not become dependent on the amount of "null" padding; and 2) no probability mass is taken from the other states of Z as is the case with the special termination symbol # in the original RY model. We found empirically that the use of either a null or an end state hurts performance to a small but significant degree.

In the last frame, two new nodes make their appearance. $send$ and $tend$ ensure we are *at or past* the end of the two strings (the RV end only checks that we are past the end). That is why $send$ depends on both a and Z. If $a > m$, $send$ (observed to be 1) is 1 with probability 1. If $a < m$, then $P(send{=}1) = 0$ and the whole sequence Z_1^ℓ has zero probability. If $a = m$, then $send$ only gets probability greater than zero if Z is not an insertion. This ensures the last source symbol is indeed consumed.

Note that we can obtain the equivalent of the total edit distance cost by using *Viterbi inference* and adding a $cost_i$ variable as a deterministic child of the random variable Z_i : in each frame the cost is equal to $cost_{i-1}$ plus 0 when Z_i is an identity, or plus 1 otherwise.

3.2 Context-dependent Model

Adding context dependence in the DBN framework is quite natural. In fig. 2, we add edges from s_i, $sprev_i$, and $snext_i$ to Z_i. The sc node is no longer required because we can enforce the consistency constraint via the CPT of Z given its parents. $snext_i$ is an RV whose value is set to the symbol at the $a_i{+}1$ position of the string, i.e., $snext_i{=}s_{a_i+1}$. Likewise, $sprev_i = s_{a_i-1}$. The Bahl model (1975) uses a dependency on s_i only. Note that s_{i-1} is not necessarily equal to s_{a_i-1}. Conditioning on s_{i-1} induces an

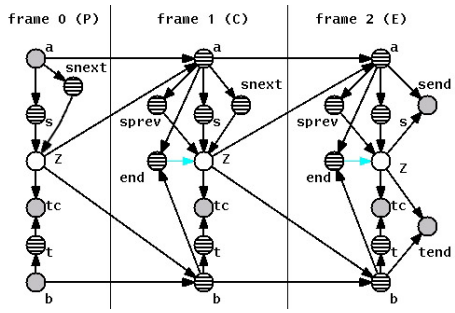

Figure 2: *Context-dependent model.*

indirect dependence on whether there was an insertion in the previous step because $s_{i-1} = s_i$ might be correlated with the event $Z_{i-1}^{(s)} = \epsilon$.

3.3 Memory Model

Memory models are another easy extension of the basic model as fig. 3 shows. Depending on whether the variable H_{i-1} linking Z_{i-1} to Z_i is stochastic or deterministic, there are several models that can be implemented; for example, a latent factor memory model when H is stochastic. The cardinality of H determines how much the information from one frame to the other is "summarized." With a deterministic implementation, we can, for example, specify the usual $P(Z_i|Z_{i-1})$ memory model when H is a simple copy of Z or have Z_i depend on the type of edit operation in the previous frame.

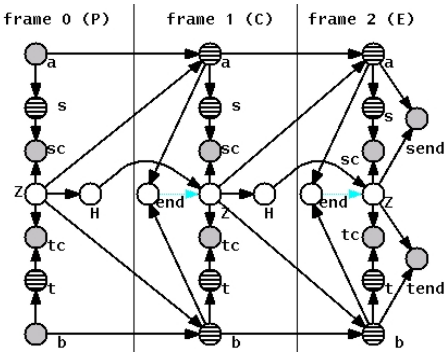

Figure 3: *Memory model. Depending on the type of dependency between Z_i and H_i, the model can be latent variable based or it can implement a deterministic dependency on a function of Z_i*

3.4 Direct Model

The *direct model* in fig. 4 is patterned on the classic HMM, where the unrolled length of graph is the same as the length of the sequence of observations. The key feature of this model is that we are required

to consume a target symbol per frame. To achieve that, we introduce two RVs, ins, with cardinality 2, and del, with cardinality at most m. The dependency of del on ins is to ensure the two events never happen concomitantly. At each frame, a is incremented either by the value of del in the case of a (possibly block) deletion or by zero or one depending on whether there was an insertion in the previous frame. An insertion also forces s to take value ϵ.

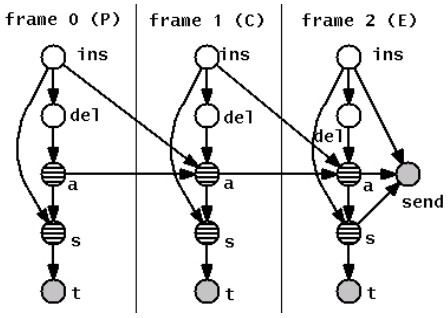

Figure 4: *Direct model.*

In essence the direct model is not very different from the context-dependent model in that here too we learn the conditional probabilities $P(t_i|s_i)$ (which are implicit in the CD model).

3.5 Length Model

While this model (fig. 5) is more complex than the previous ones, much of the network structure is "control logic" necessary to simulate variable length-unrolling of the graph template. The key idea is that we have a new stochastic hidden RV, $inclen$, whose value added to that of the RV $inilen$ determines the number of edit operations we are allowed. A counter variable, $counter$ is used to keep track of the frame number and when the required number is reached, the RV $atReqLen$ is triggered. If at that point we have just reached the end of one of the strings while the end of the other one is reached in this frame or a previous one, then the variable end is *explained* (it has positive probability). Otherwise, the entire sequence of edit operations up to that point has zero probability.

4 Pronunciation Classification

In pronunciation classification we are given a *lexicon*, which consists of words and their corresponding *canonical pronunciations*. We are also provided with *surface pronunciations* and asked to find the most likely corresponding words. Formally, for each

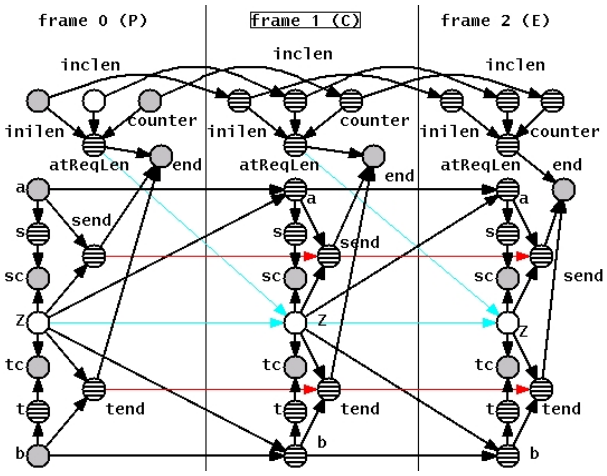

Figure 5: *Length unrolling model.*

surface form, t_1^n, we need to find the set of words $\hat{W}$ s.t. $\hat{W} = argmax_w P(w|t_1^n)$. There are several ways we could model the probability $P(w|t_1^n)$. One way is to assume a generative model whereby a word w and a surface pronunciation t_1^n are related via an underlying canonical pronunciation s_1^m of w and a stochastic process that explains the transformation from s_1^m to t_1^n. This is summarized in equation 3. $C(w)$ denotes the set of canonical pronunciations of w.

$$\hat{W} = \underset{w}{\text{argmax}} \sum_{s_1^m \in C(w)} P(w|s_1^m) P(s_1^m, t_1^n) \qquad (3)$$

If we assume uniform probabilities $P(w|s_1^m)$ ($s_1^m \in C(w)$) and use the max approximation in place of the sum in eq. 3 our classification rule becomes

$$\hat{W} = \{w | \hat{S} \cap C(w) \neq \emptyset, \hat{S} = \underset{s_1^m}{\text{argmax}}\ P(s_1^m, t_1^n)\} \quad (4)$$

It is straightforward to create a DBN to model the joint probability $P(w, s_1^m, t_1^n)$ by adding a word RV and a canonical pronunciation RV on top of any of the previous models.

There are other pronunciation classification approaches with various emphases. For example, Rentzepopoulos and Kokkinakis (1996) use HMMs to convert phoneme sequences to their most likely orthographic forms in the absence of a lexicon.

4.1 Data

We use Switchboard data (Godfrey et al., 1992) that has been hand annotated in the context of the Speech Transcription Project (STP) described in (Greenberg et al., 1996). Switchboard consists of spontaneous informal conversations recorded over the

phone. Because of the informal non-scripted nature of the speech and the variety of speakers, the corpus presents much variety in word pronunciations, which can significantly deviate from the prototypical pronunciations found in a lexicon. Another source of pronunciation variability is the noise introduced during the annotation of speech segments. Even when the phone labels are mostly accurate, the start and end time information is not as precise and it affects how boundary phones get aligned to the word sequence. As a reference pronunciation dictionary we use a lexicon of the 2002 Switchboard speech recognition evaluation. The lexicon contains 40000 entries, but we report results on a reduced dictionary[5] with 5000 entries corresponding to only those words that appear in our train and test sets. Ristad and Yianilos use a few additional lexicons, some of which are corpus-derived. We did reproduce their results on the different types of lexicons.

For testing we randomly divided STP data into 9495 training words (corresponding to 9545 pronunciations) and 912 test words (901 pronunciations). For the Levenshtein and MCI results only, we performed ten-fold cross validation to verify we did not pick a non-representative test set. Our models are implemented using GMTK, a general-purpose DBN tool originally created to explore different speech recognition models (Bilmes and Zweig, 2002). As a sanity check, we also implemented the MCI model in C following RY's algorithm.

The error rate is computed by calculating, for each pronunciation form, the fraction of words that are correctly hypothesized and averaging over the test set. For example if the classifier returns five words for a given pronunciation, and two of the words are correct, the error rate is 3/5*100%.

Three EM iterations are used for training. Additional iterations overtrained our models.

4.2 Results

Table 1 summarizes our results using DBN based models. The basic MCI model does marginally better than the Levenshtein edit distance. This is consistent with the finding in RY: their gains come from the joint learning of the probabilities $P(w|s_1^m)$ and $P(s_1^m, t_1^n)$. Specifically, the word model accounts for much of their gains over the Levenshtein distance. We use uniform priors and the simple classification rule in eq. 4. We feel it is more compelling that we are able to significantly improve upon standard edit distance and the MCI model without using any lexicon or word model.

Memory Models Performance improves with the addition of a direct dependence of Z_i on Z_{i-1}. The biggest improvement (27.65% ER) however comes from conditioning on $Z_{i-1}^{(t)}$, the target symbol that is hypothesized in the previous step. There was no gain when conditioning on the type of edit operation in the previous frame.

Context Models Interestingly, the exact opposite from the memory models is happening here when we condition on the source context (versus conditioning on the target context). Conditioning on s_i gets us to 21.70%. With s_i, s_{i-1} we can further reduce the error rate to 20.26%. However, when we add a third dependency, the error rate worsens to 29.32%, which indicates a number of parameters too high for the given amount of training data. Backoff, interpolation, or state clustering might all be appropriate strategies here.

Position Models Because in the previous models, when conditioning on the past, boundary conditions dictate that we use a different CPT in the first frame, it is fair to wonder whether part of the gain we witness is due to the implicit dependence on the source-target string position. The (small) improvement due to conditioning on b_i indicates there is such dependence. Also, the fact that the target position is more informative than the source one is likely due to the misalignments we observed in the phonetically transcribed corpus, whereby the first or last phones would incorrectly be aligned with the previous or next word resp. I.e., the model might be learning to not put much faith in the start and end positions of the target string, and thus it boosts deletion and insertion probabilities at those positions. We have also conditioned on coarser-grained positions (beginning, middle, and end of string) but obtained the same results as with the fine-grained dependency.

Length Models Modeling length helps to a small extent when it is added to the MCI and MCD models. Belying the assumption motivating this model, we found that the distribution over the RV $inclen$ (which controls how much the edit sequence extends

[5] Equivalent to the *E2* lexicon in RY.

beyond the length of the source string) is skewed towards small values of $inclen$. This indicates on that insertions are rare when the source string is longer than the target one and vice-versa for deletions.

Direct Model The low error rate obtained by this model reflects its similarity to the context-dependent model. From the two sets of results, it is clear that source string context plays a crucial role in predicting canonical pronunciations from corpus ones. We would expect additional gains from modeling context dependencies across time here as well.

Model	Z_i Dependencies	% Err rate
Lev	none	**35.97**
Baseline	none	**35.55**
Memory	Z_{i-1}	30.05
	editOperationType(Z_{i-1})	36.16
	stochastic binary H_{i-1}	33.87
	$Z^{(s)}_{i-1}$	29.62
	$Z^{(t)}_{i-1}$	**27.65**
Context	s_i	21.70
	t_i	32.06
	s_i, s_{i-1}	**20.26**
	t_i, t_{i-1}	28.21
	s_i, s_{i-1}, s_{a_i+1}	29.32
	s_i, s_{a_i+1} (s_{a_i-1} in last frame)	23.14
	s_i, s_{a_i-1} (s_{a_i+1} in first frame)	23.15
Position	a_i	33.80
	b_i	**31.06**
	a_i, b_i	34.17
Mixed	b_i,s_i	**22.22**
	$Z^{(t)}_{i-1}$,s_i	24.26
Length	none	33.56
	s_i	**20.03**
Direct	none	**23.70**

Table 1: *DBN based model results summary.*

When we combine the best position-dependent or memory models with the context-dependent one, the error rate decreases (from 31.31% to 25.25% when conditioning on b_i and s_i; and from 28.28% to 25.75% when conditioning on $z^{(t)}_{i-1}$ and s_i) but not to the extent conditioning on s_i alone decreases error rate. Not shown in table 1, we also tried several other models, which although they are able to produce reasonable alignments (in the sense that the Levenshtein distance would result in similar alignments) between two given strings, they have extremely poor discriminative ability and result in error rates higher than 90%. One such example is a model in which Z_i depends on both s_i and t_i. It is easy to see where the problem lies with this model once one considers that two very different strings might still get a higher likelihood than more similar pair because, given s and t s.t. $s \neq t$, the probability of identity is obviously zero and that of insertion or deletion can be quite high; and when $s = t$, the probability of insertion (or deletion) is still positive. We observe the same non-discriminative behavior when we replace, in the MCI model, Z_i with a hidden RV X_i, where X_i takes as values one of the four edit operations.

5 Computational Considerations

The computational complexity of inference in a graphical model is related to the state space of the largest clique (maximal complete subgraph) in the graph. In general, finding the smallest such clique is NP-complete (Arnborg et al., 1987).

In the case of the MCI model, however, it is not difficult to show that the smallest such clique contains all the RVs within a frame and the complexity of doing inference is order $O(mn \cdot max(m, n))$. The reason there is a complexity gap is that the source and target position variables are indexed by the frame number and we do not exploit the fact that even though we arrive at a given source-target position pair along different edit sequence paths at different frames, the position pair is really the same regardless of its frame index. We are investigating generic ways of exploiting this constraint.

In practice, however, state space pruning can significantly reduce the running time of DBN inference. Ukkonen (1985) reduces the complexity of the classic edit distance to $O(d \cdot max(m, n))$, where d is the edit distance. The intuition there is that, assuming a small edit distance, the most likely alignments are such that the source position does not diverge too much from the target position. The same intuition holds in our case: if the source and the target position do not get too far out of sync, then at each step, only a small fraction of the $m \cdot n$ possible source-target position configurations need be considered.

The direct model, for example, is quite fast in practice because we can restrict the cardinality of the *del* RV to a constant c (i.e. we disallow long-span deletions, which for certain applications is a reasonable restriction) and make inference linear in n with a running time constant proportional to c^2.

6 Conclusion

We have shown how the problem of learning edit distance costs from data can be modeled quite naturally using Dynamic Bayesian Networks even though the problem lacks the temporal or order constraints that other problems such as speech recognition exhibit. This gives us confidence that other important problems such as machine translation can benefit from a Graphical Models perspective. Machine translation presents a fresh set of challenges because of the large combinatorial space of possible alignments between the source string and the target.

There are several extensions to this work that we intend to implement or have already obtained preliminary results on. One is simple and block transposition. Another natural extension is modeling edit distance of multiple strings.

It is also evident from the large number of dependency structures that were explored that our learning algorithm would benefit from a structure learning procedure. Maximum likelihood optimization might, however, not be appropriate in this case, as exemplified by the failure of some models to discriminate between different pronunciations. Discriminative methods have been used with significant success in training HMMs. Edit distance learning could benefit from similar methods.

References

S. Arnborg, D. G. Corneil, and A. Proskurowski. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284.

L. R. Bahl and F. Jelinek. 1975. Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition. *Trans. on Information Theory*, 21:404–411.

J. Bilmes and C. Bartels. 2003. On triangulating dynamic graphical models. In *Uncertainty in Artificial Intelligence: Proceedings of the 19th Conference*, pages 47–56. Morgan Kaufmann.

J. Bilmes and G. Zweig. 2002. The Graphical Models Toolkit: An open source software system for speech and time-series processing. *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*.

J. J. Godfrey, E. C. Holliman, and J. McDaniel. 1992. SWITCHBOARD: Telephone speech corpus for research and development. In *ICASSP*, volume 1, pages 517–520.

S. Greenberg, J. Hollenback, and D. Ellis. 1996. Insights into spoken language gleaned from phonetic transcription of the switchboard corpus. In *ICSLP*, pages S24–27.

P. N. Klein. 1998. Computing the edit-distance between unrooted ordered trees. In *Proceedings of 6th Annual European Symposium*, number 1461, pages 91–102.

S.L. Lauritzen. 1996. *Graphical Models*. Oxford Science Publications.

G. Leusch, N. Ueffing, and H. Ney. 2003. A novel string-to-string distance measure with applications to machine translation evaluation. In *Machine Translation Summit IX*, pages 240–247.

V. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Dokl.*, 10:707–710.

R. Lowrance and R. A. Wagner. 1975. An extension to the string-to-string correction problem. *J. ACM*, 22(2):177–183.

M. Mohri. 2002. Edit-distance of weighted automata. In *CIAA*, volume 2608 of *Lecture Notes in Computer Science*, pages 1–23. Springer.

K. Murphy. 2002. *Dynamic Bayesian Networks: Representation, Inference and Learning*. Ph.D. thesis, U.C. Berkeley, Dept. of EECS, CS Division.

R. Myers, R.C. Wison, and E.R. Hancock. 2000. Bayesian graph edit distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22:628–635.

M. Neuhaus and H. Bunke. 2004. A probabilistic approach to learning costs for graph edit distance. In *ICPR*, volume 3, pages 389–393.

P. A. Rentzepopoulos and G. K. Kokkinakis. 1996. Efficient multilingual phoneme-to-grapheme conversion based on hmm. *Comput. Linguist.*, 22(3):351–376.

E. S. Ristad and P. N. Yianilos. 1998. Learning string edit distance. *Trans. on Pattern Recognition and Machine Intelligence*, 20(5):522–532.

D. Shapira and J. A. Storer. 2003. Large edit distance with multiple block operations. In *SPIRE*, volume 2857 of *Lecture Notes in Computer Science*, pages 369–377. Springer.

E. Ukkonen. 1985. Algorithms for approximate string matching. *Inf. Control*, 64(1-3):100–118.

R. A. Wagner and M. J. Fischer. 1974. The string-to-string correction problem. *J. ACM*, 21(1):168–173.

J. Wei. 2004. Markov edit distance. *Trans. on Pattern Analysis and Machine Intelligence*, 26(3):311–321.