Minimum Bayes Risk Decoding and System Combination Based on a Recursion for Edit Distance

Haihua Xu[a], Daniel Povey[b], Lidia Mangu[c], Jie Zhu[a]

[a]*Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, 200240, China*
[b]*Microsoft Research, Redmond, WA, USA*
[c]*IBM T.J. Watson Research Center, Yorktown Heights, NY, USA*

Abstract

In this paper we describe a method that can be used for Minimum Bayes Risk (MBR) decoding for speech recognition. Our algorithm can take as input either a single lattice, or multiple lattices for system combination. It has similar functionality to the widely used Consensus method, but has a clearer theoretical basis and appears to give better results both for MBR decoding and system combination. Many different approximations have been described to solve the MBR decoding problem, which is very difficult from an optimization point of view. Our proposed method solves the problem through a novel forward-backwards recursion on the lattice, not requiring time markings. We prove that our algorithm iteratively improves a bound on the Bayes Risk.

Keywords: Speech Recognition, Minimum Bayes Risk Decoding

1. Introduction

Speech recognition systems generally make use of the Maximum A Posteriori (MAP) decoding rule:

$$\begin{aligned} W^* &= \mathbf{argmax}_W P(W|\mathcal{X}) \\ &= \mathbf{argmax}_W P(W)p(\mathcal{X}|W), \end{aligned} \tag{1}$$

Email addresses: haihua_xu@sjtu.edu.cn (Haihua Xu), dpovey@microsoft.com (Daniel Povey), mangu@us.ibm.com (Lidia Mangu), zhujie@sjtu.edu.cn (Jie Zhu)

where W is the word-sequence, $\mathcal{X}$ is the acoustic observation sequence, $P(W)$ is the language model probability and $p(\mathcal{X}|W)$ is the acoustic likelihood (ignoring likelihood scaling for now).

It can be shown that under assumptions of model correctness, Equation (1) gives the Minimum Bayes Risk estimate with respect to the *sentence* error, i.e. it minimizes the probability of choosing the wrong sentence. It is not clear that this is the best approach; the standard error metric for speech recognition systems is the Word Error Rate (WER), computed for sentences $n = 1 \ldots N$ as:

$$\mathrm{WER}(W_1 \ldots W_N | R_1 \ldots R_N) = \frac{\sum_{n=1}^{N} L(W_n, R_n)}{\sum_{n=1}^{N} |R_n|}, \tag{2}$$

where $L(A, B)$ is the Levenshtein edit distance [13] between sequence A and B, W_n and R_n are the n'th transcribed sentence and reference sentence respectively, and $|A|$ is the number of symbols in sequence A. WER is usually expressed as a percentage.

A substantial amount of work has previously been done on decoding methods that minimize a WER-like risk measure, based on a lattice of alternative outputs from a speech recognizer. These methods all fall under the general category of Minimum Bayes Risk (MBR) decoding. Note that MBR decoding is an ambiguous term because it is defined only with respect to a particular measure of risk. For speech recognition (including this paper) we generally have in mind the Levenshtein edit distance, but in the machine translation literature, N-gram counting methods related to the BLEU score [15] are generally used. In this paper we introduce a technique for MBR decoding (w.r.t. the Levenshtein edit distance) that is simpler and has a clearer theoretical basis than the most widely used method, known as Consensus [12]. The core of it is a two-dimensional recursion that in one dimension is like a forwards-backwards algorithm on a lattice and in the other is like the Levenshtein edit distance recursion.

In Section 2 we introduce the concept of Minimum Bayes Risk decoding and describe previous work in this area. In Section 3 we give a more detailed overview of our approach and describe how it relates to previous work. In Section 4 we describe the Levenshtein edit distance and give an algorithm to compute it, which will motivate our lattice-based algorithm. In Section 5 we discuss lattices and introduce our notation for them. In Section 6 we describe our method for approximating the edit distance between a lattice and a word sequence. In Section 7 we explain how we optimize our hypothesis with respect to this metric. In Section 8 we describe our experimental setup, and in Section 9 we present our experiments. In Section 10 we conclude. In Appendix A we prove that our approximated edit distance is an upper bound on the true edit distance, and in Appendix B we prove that our algorithm decreases the approximated edit distance on each iteration until convergence. Appendix C describes extensions of the algorithm to handle alternative lattice formats and to compute time alignments.

2. Minimum Bayes Risk Decoding Methods

The Bayes Risk with respect to the Levenshtein distance may be written as:

$$\mathcal{R}(W) = \sum_{W'} P(W'|\mathcal{X}) L(W, W'), \tag{3}$$

and minimizing this is equivalent to minimizing the expected Word Error Rate (given the assumption of model correctness). The general aim of Minimum Bayes Risk (MBR) decoding methods is to compute the W that minimizes (3) as exactly as possible, i.e. to compute

$$W^* = \mathbf{argmin}_W \sum_{W'} P(W'|\mathcal{X}) L(W, W') \tag{4}$$

where the values of W and W' are generally constrained to a finite set covered by an N-best sentence list or a lattice. In order to motivate the problem at this point, we will give a simple example where the output differs between this method and the MAP formula (see Figure 1). Figure 1(a) shows the probabilities assigned by our model to different sentences; in this example it only assigns nonzero probabilities to three different sentences. Figure 1(b) shows the expected sentence-level error metric and word-level error metric respectively, if we output the string given in the corresponding table row, and assuming the modeled probabilities are accurate. There is no reference sentence here; the assumption is that our model accurately models ambiguity in the data. The check mark in each column is next to the hypothesis with the lowest "expected error" for that metric; this would be the output of the corresponding decoding algorithm. The MBR estimate "A D C" has a lower expected word error, but a higher expected sentence error, than the MAP estimate "A B C" (which corresponds to minimizing the expected sentence-level error rate). In general MBR decoding will increase the sentence error

Figure 1: Example of MBR estimate differing from MAP estimate

Sentence	Model Prob
A B C	0.4
A D X	0.3
A D Y	0.3

(a) Modeled probabilities

Decoded Sentence	Expected Errors	
	Sent	Word
A B C	0.6✓	1.2
A D X	0.7	1.1
A D Y	0.7	1.1
A D C	1.0	1.0 ✓

(b) Expected errors

rate, since it exploits the difference between string-level and word-level error rates. This should be borne in mind when evaluating the appropriateness of MBR decoding to a particular situation, as one runs the risk of "over-tuning" to a particular metric such as Word Error Rate, which may only be a proxy for an underlying risk that is of interest. Something else to note in this example is that the utterance chosen by MBR decoding is assigned zero probability by the model.

The idea of Minimum Bayes Risk decoding with respect to the edit distance was introduced in [21]. In that paper, both the $\sum$ and the **argmin** of Equation (4) were limited to a long N-best list representing the most likely sentences for the utterance, and it was observed that the **argmin** could be chosen from a relatively short N-best list since the one chosen was generally in the top 10. The edit distance was computed using the standard quadratic-time algorithm. However, this is not a practical or general solution since the length of N-best list required for the technique to perform well is expected to rise exponentially with the length of the utterance.

Soon afterwards, two different workable approximations to the MBR decoding problem were published [6, 12], of which [12] (Consensus, a.k.a. Confusion Network Decoding) has become widely used. In this technique, the lattice is compressed into structure called a "confusion network", or informally, a "sausage string". A confusion network has a fixed number of word positions, and at each position there are a number of alternative words (or the special symbol ϵ meaning no word is present). It is possible to show that under reasonable assumptions, the MBR estimate given the confusion network is obtained by taking the most probable symbol (or ϵ, meaning no symbol) at each position.

Shortly afterwards, the Confusion Network idea was applied to system combination, with a technique known as Confusion Network Combination (CNC) [4]. This has become widely used as a replacement for ROVER [5], which is a method to take the output of multiple systems and produce a combined output sequence by a procedure in which "voting" occurs at each word position. CNC has the same functionality as ROVER but operates on confusion networks rather than words; it operates by replacing the word alignment used to align different sequences in ROVER, with a similar procedure operating on confusion network positions. This is used to produce a combined confusion network before picking the most likely word at each position.

Now we summarize some more recent but less well known approaches. In [23] a frame-weighted version of a word error metric is minimized, using the time information in the lattice to provide the alignment. Improvements were reported versus the standard decoding method, but the method was not compared with Consensus. Our own previous work [25] uses similar approximations to those used in Minimum Phone Error (MPE) and Minimum Word Error (MWE) discriminative training [18] to compute the MBR estimate. Results appeared to be slightly better than Consensus. Another recent paper [10] also used various MPE/MWE types of approximations, but also used a Confusion Network type of structure as in Consensus; none of the proposed methods seemed to perform quite as well as Consensus. An interesting piece of work on a slightly different problem is [8], which aimed to reduce the approximations involved in MPE/MWE training by using Weighted Finite State Transducer (WFST) techniques [14]. This is an extension of the problem of computing $\sum_{W'} P(W'|\mathcal{X})L(W, W')$ exactly without approximations, but we do not see any obvious extensions of this approach to the case of MBR decoding. We also note our recent conference paper [24] which reports an earlier version of the work presented here; that algorithm differs slightly from the current one (that paper optimizes a less tight bound on the edit distance, due to a difference in the recursion formula).

3. Our Method

We now summarize our approach. Like other Minimum Bayes Risk decoding approaches such as Consensus [12], we aim to evaluate Equation (4) as exactly as possible. What this means is, given a lattice of alternative hypotheses, we aim to produce the sentence that minimizes the Bayes Risk as closely as possible. This is a very difficult optimization problem. The difficulty is largely due to the non-local nature of the Levenshtein edit distance. When more "local" risk measures based on counting N-grams are used, as they are in the Machine Translation literature, efficient exact solutions may be obtained, e.g. see [7]. When the Levenshtein edit distance is the risk measure, a solution to the decoding problem is only easy to describe when working with N-best sentence lists; however, the length of such a list required to cover all word confusions with a particular cutoff of word likelihood is expected to grow exponentially with the utterance length, which leads to naïve algorithms taking exponential time[1]. Previous approaches such as Consensus made various approximations in order to obtain an efficient solution. We also make approximations, but we believe that our approximations are much less severe than those of competing approaches. In fact we are able to argue rigorously about our approach and demonstrate that we are minimizing an upper bound on the Bayes Risk. This is not possible for most competing methods.

We note that our approach is formulated as a solution for the Minimum Bayes Risk decoding problem, which applies to a single lattice of alternatives. However, it naturally leads to a method for system combination (we describe this extension in Section 7.4). This kind of extension is also possible for other Minimum Bayes Risk decoding methods such as Consensus, which can be extended via Confusion Network Combination (CNC) [4]; however, whereas CNC is a nontrivial extension of Consensus, our system combination method is a very simple extension of our lattice-based decoding algorithm.

We present an algorithm for approximating the Bayes Risk of (3) given a lattice and a hypothesis. This is essentially a hybrid of two algorithms: the standard quadratic-time algorithm used to compute the Levenshtein distance; and the forward-backward algorithm for computing probabilities of traversing lattice arcs. We prove that the approximated lattice edit distance $\hat{L}$ which we compute in our algorithm is an upper bound on the true edit distance averaged over the paths in the lattice. We then build on this algorithm to obtain an optimization process that modifies the hypothesis to iteratively

[1]The statement that the length of N-best list grows exponentially needs to be interpreted correctly. It is true if we assume that the N-best list must cover all sentences such that each word in that sentence has a posterior greater than some cutoff.

improve the approximated Bayes Risk $\hat{L}$. We prove that our optimization process is guaranteed to decrease $\hat{L}$ on each step (until convergence).

4. Levenshtein edit distance

The Levenshtein distance (edit distance) between two sequences of symbols W and X is written as:

$$L(W, X) \tag{5}$$

where W and X are sequences, e.g. $W = (w_1, w_2, \ldots, w_N)$, not necessarily of the same length. In this paper we enclose sequences in parentheses. We write the length of a sequence W as $|W|$. The edit distance is defined as the minimum number of symbols inserted, deleted and substituted in W versus X (or vice versa), taken over any possible alignment between the pair of sequences. We do not formally define the concept of sequence alignment here as it is quite obvious, but we show an example: for the symbol sequences `A B C D` and `X C D E`, an optimal alignment would be: $\begin{array}{ccccc} \texttt{A} & \texttt{B} & \texttt{C} & \texttt{D} & \epsilon \\ \texttt{X} & \epsilon & \texttt{C} & \texttt{D} & \texttt{E} \end{array}$, where we use ϵ to mean "no symbol". We formulate the edit-distance computation as a dynamic-programming recursion using an array of partial edit distances of size $|A|$+1 by $|B|$+1, representing all possible initial sub-strings of A and B. We define an edit distance on pairs of individual symbols, as:

$$l(a, b) = \begin{cases} 0, & a = b \\ 1, & a \neq b \end{cases} \tag{6}$$

where a and b may be either word symbols or ϵ. The string-level recursion is as follows, where we use a_n for the $n'th$ element of A (with $1 \leq n \leq |A|$):

$$L(A, B) = \min \left\{ \begin{array}{c} l(a_{|A|}, b_{|B|}) + L(A_1^{|A|-1}, B_1^{|B|-1}) \\ l(a_{|A|}, \epsilon) + L(A_1^{|A|-1}, B) \\ l(\epsilon, b_{|B|}) + L(A, B_1^{|B|-1}) \end{array} \right\} \tag{7}$$

where to simplify notation we assume that invalid choices are never taken, e.g. the top two choices are not valid if A is the empty sequence. To clarify the notation, $A_1^{|A|-1}$ means all but the last symbol of A. The base case of the recursion on strings is:

$$L(\emptyset, \emptyset) = 0, \tag{8}$$

```
L = Levenshtein(A, B):
1: Initialize an array $\alpha(0 \ldots |A|, 0 \ldots |B|)$.
2: for $q \leftarrow 0 \ldots |A|$ do
3:     for $r \leftarrow 0 \ldots |B|$ do
4:         if $q = 0 \wedge r = 0$ then
5:             $\alpha(q,r) \leftarrow 0$
6:         else
7:             $\alpha(q,r) \leftarrow \min \left\{ \begin{array}{c} \alpha(q{-}1,r) + l(a_q,\epsilon) \\ \alpha(q{-}1,r{-}1) + l(a_q,b_r) \\ \alpha(q,r{-}1) + l(\epsilon,b_r) \end{array} \right\}$
8:         end if
9:     end for
10: end for
11: $L \leftarrow \alpha(|A|,|B|)$
```

Figure 2: Recursive Levenshtein edit distance computation

where we use $\emptyset$ to denote the empty sequence. Note that we have formulated this in such a way that we may insert an arbitrary number of ϵ symbols into the sequences A and B and this will not affect the edit distance between them. This is important for our algorithms. Figure 2 shows how we would compute the edit distance using dynamic programming. It is to be understood that options in the min expression on Line 7 that would result in accessing an out-of-bounds array element are not taken.

5. Lattices

In speech recognition, the term *lattice* generally refers to a representation of alternative speech recognizer outputs. For our purposes we can treat a lattice as a directed acyclic graph with unique begin and end points, i.e. a start node and an end node. Note that some common lattice formats (e.g. HTK [26]) and finite-state transducer libraries [1] allow multiple end nodes, and in Appendix C we explain how to extend our algorithms to that case.

For purposes of the current paper, we will say that formally a lattice $\mathcal{L}$ has nodes $n \in \mathcal{L}$, and we write $b(\mathcal{L})$ and $e(\mathcal{L})$ for the start and end nodes respectively which we assume to be unique. We assume that we can access these nodes in forwards or backwards topological order. Each node n has a set of outgoing arcs post(n) and a set of incoming arcs pre(n). The starting node of an arc is written s(a) and the ending node is written e(a). The likelihood on an arc a is written $p(a)$ (this would be the product of the appropriately scaled acoustic and language model likelihoods; see (16) for an example). We write $w(a)$ for the word label of an arc. This may be the special symbol ϵ; we map non-scored words such as silence or noise to ϵ in the lattice. In algorithms, to simplify the notation we will identify the nodes with the integers $1 \ldots N$, where $N = |\mathcal{L}|$, and we assume they are in topological order (hence $b(\mathcal{L}) = 1$ and $e(\mathcal{L}) = N$).

We have not defined exactly what the lattice should represent in terms of the alternative sentences that could have generated an utterance. Lattices are frequently evaluated in terms of the lattice Word Error Rate (lattice WER, also known as lattice oracle error rate), which is the average error rate of the best path through each lattice, measured over a set of utterances and their corresponding lattices. For some algorithms (e.g. rescoring a lattice with a different acoustic model), a low lattice WER is sufficient to ensure good results. However, for algorithms such as ours in which the posterior probabilities within the lattice are used, the lattice needs to have additional properties: it is important that the posterior probability of any sentence W given the acoustic and language models is closely approximated by its posterior probability within the lattice. We could express this as follows (ignoring probability scaling factors):

$$\frac{P(W)p(\mathcal{X}|W)}{\sum_{W'} P(W')p(\mathcal{X}|W')} \simeq P(W|\mathcal{L}), \tag{9}$$

where $\mathcal{X}$ is the acoustic observations, $P(W)$ is the language model, $p(\mathcal{X}|W)$ is the acoustic likelihood, and $P(W|\mathcal{L})$ is the posterior probability of W obtained by the forward-backward algorithm over the lattice. Equation 9 cannot be an exact equality in general, because the lattice will typically not contain all possible sentences W with nonzero probability. The easiest way to ensure Equation 9 is satisfied (up to some threshold) is to ensure that the lattice contains all possible sentences within some likelihood beam of the most likely one (but each sentence exactly once), and has accurate likelihoods on its arcs. However, for backoff language models it can be difficult to ensure that each sentence is represented exactly once due to duplicate paths through "backoff" and non-backoff language model states. One possible way to solves this problem is to use decoding graphs in which the "higher order" arcs represent only the "extra part" of the language model probability; we are not aware that this has ever been done. In the work reported here we used

```
Forwards-Backwards($\mathcal{L}$):
 1: $N \leftarrow |\mathcal{L}|$
 2: Initialize arrays $\alpha(1 \dots N)$, $\beta(1 \dots N)$  // Store as log
 3: $\alpha(1) \leftarrow 1$, $\beta(N) \leftarrow 1$.
 4: for $n \leftarrow 2 \dots N$ do
 5:     $\alpha(n) \leftarrow \sum_{a \in \mathrm{pre}(n)} \alpha(\mathrm{s}(a)) p(a)$
 6: end for
 7: for $n \leftarrow N{-}1 \dots 1$ do
 8:     $\beta(n) \leftarrow \sum_{a \in \mathrm{post}(n)} \beta(\mathrm{e}(a)) p(a)$
 9: end for
10: for arcs $a$ in lattice $\mathcal{L}$ do
11:     $\gamma(a) \leftarrow \frac{\alpha(\mathrm{s}(a))\beta(\mathrm{e}(a))p(a)}{\alpha(N)}$  // Do something with this
12: end for
```

Figure 3: Forwards-Backwards likelihood computation on a lattice

the available lattices "as-is" without evaluating how closely Equation 9 is satisfied; we merely wish to emphasize that the lattice quality may affect the results of our algorithm and other algorithms that use lattices, in a way that is not fully captured by the lattice WER metric.

For an example of an algorithm on a lattice, and to clarify our notation, Figure 3 shows how we would compute occupation probabilities $0 < \gamma(a) \leq 1$ for arcs a in the lattice.

6. Edit distance calculation on lattices

In this section we introduce an edit distance calculation on lattices. It computes an upper bound on the Levenshtein edit distance averaged over a lattice. We will use the notation $S \in \mathcal{L}$ to represent a path S through the lattice starting from the start node $b(\mathcal{L})$ and ending at the end node; S represents a sequence of arcs $S = (s_1, \dots, s_N)$. We will write $p(S)$ as the total likelihood

$$p(S) = \prod_{n=1}^{|S|} p(s_n) \tag{10}$$

and $W(S)$ as the sequence of symbols

$$W(S) = (w(s_1), w(s_2), \dots, w(s_{|S|})). \tag{11}$$

We define the exact edit distance between a string R and a lattice $\mathcal{L}$ as:

$$L(\mathcal{L}, R) \quad = \quad \frac{\sum_{S \in \mathcal{L}} p(S) L(W(S), R)}{\sum_{S \in \mathcal{L}} p(S)}, \tag{12}$$

and this is the same as the Bayes Risk of (3), except approximating all word sequences by those present in the lattice (and assuming that sentences each appear exactly once in the lattice and the probabilities on the lattice arcs are accurate). We need to minimize (12) with respect to R. The first step is to calculate the value of (12) given a fixed R. Doing this exactly and efficiently is hard (although it is possible, see [8]), and our algorithm computes an upper bound $\hat{L}(\mathcal{L}, R) \geq L(\mathcal{L}, R)$ (proved in Appendix A). We believe that it is a very close upper bound in practice; see Section 9.4 for experiments relating to this.

Our edit distance computation of Figure 4 is essentially a combination of the lattice forward-backward algorithm of Figure 3 and the recursive edit distance computation of Figure 2. Note on Line 17, a reference to δ. This is a small positive constant $0 < \delta \ll 1$, which we introduce to break a symmetry; we will explain in the next section why it is needed.

7. Word sequence optimization

7.1. Statistics computation

In Figure 5 we extend the algorithm described above to accumulate statistics that will allow us to optimize the sequence R. This will later be used in an iterative process where R changes from iteration to iteration. We compute the quantity $\gamma(q, s)$, which is the probability with which symbol (or ϵ) s aligned with position q in the word sequence (where $1 \leq q \leq |R|$). The statistics have the property that $\sum_s \gamma(q, s) = 1$. We assume that $\gamma(q, s)$ would be stored in a data structure that makes use of the sparsity of the statistics (i.e. not an array of dimension $|R|$ times the number of distinct words in the lattice).

7.2. Optimization algorithm

The overall algorithm in which we optimize the sequence R is given in Figure 6. It essentially consists of picking the most likely symbol at each word position on each iteration. We prove in Appendix B that each step (before convergence) decreases our approximated edit distance $\hat{L}$. A key feature is

```
$\hat{L} = \text{Edit-Distance}(\mathcal{L}, R)$:
1: $N \leftarrow |\mathcal{L}|$, $Q \leftarrow |R|$
2: Initialize array $\alpha(1 \ldots N)$   // As in fwd-bkwd. Store as log
3: Initialize array $\alpha'(1 \ldots N, 0 \ldots Q)$   // Forward node edit dist
4: Initialize array $\alpha'_{arc}(0 \ldots Q)$   // Forward arc edit dist (temp)
5: $\alpha(1) \leftarrow 1$, $\alpha'(1, 0) \leftarrow 0$
6: for $q \leftarrow 1 \ldots Q$ do
7:     $\alpha'(1, q) \leftarrow \alpha'(1, q-1) + l(\epsilon, r_q)$
8: end for
9: for $n \leftarrow 2 \ldots N$ do
10:    $\alpha(n) \leftarrow \sum_{a \in \text{pre}(n)} \alpha(\text{s}(a)) p(a)$
11:    $\forall q, \alpha'(n, q) \leftarrow 0$
12:    for $a \in \text{pre}(n)$ do
13:       for $q \leftarrow 0 \ldots Q$ do
14:          if $q = 0$ then
15:             $\alpha'_{arc}(q) \leftarrow \alpha'(\text{s}(a), q) + l(w(a), \epsilon) + \delta$
16:          else
17:             $\alpha'_{arc}(q) \leftarrow \min \left\{ \begin{array}{c} \alpha'(\text{s}(a), q-1) + l(w(a), r_q) \\ \alpha'(\text{s}(a), q) + l(w(a), \epsilon) + \delta \\ \alpha'_{arc}(q-1) + l(\epsilon, r_q) \end{array} \right\}$
18:          end if
19:          $\alpha'(n, q) \leftarrow \alpha'(n, q) + \frac{\alpha(\text{s}(a)) p(a)}{\alpha(n)} \alpha'_{arc}(q)$
20:       end for
21:    end for
22: end for
23: $\hat{L} \leftarrow \alpha'(N, Q)$
```

Figure 4: Edit distance computation on a lattice

```
$(\hat{L}, \gamma(\cdot,\cdot)) = \text{Acc-Stats}(\mathcal{L}, R)$:
1: $N \leftarrow |\mathcal{L}|$, $Q \leftarrow |R|$
2: Initialize array $\alpha(1 \dots N)$ // Store as log
3: Initialize array $\alpha'(1 \dots N, 0 \dots Q)$ // Forward node edit dist
4: Initialize array $\alpha'_{arc}(0 \dots Q)$ // Forward arc edit dist (tmp)
5: Initialize array $\beta'(1 \dots N, 0 \dots Q)$ // Backward occ-prob
6: Initialize array $\beta'_{arc}(0 \dots Q)$ // Temporary beta for arcs.
7: Initialize array $b_{arc}(1 \dots Q)$ // Temporary back-trace $\in \{1, 2, 3\}$.
8: Initialize associative array $\gamma(1 \leq q \leq Q, s)$.
9: Do lines 5 to 23 of Fig. 4 to obtain $\alpha(n)$, $\alpha'(n, q)$.
10: $\forall n, q,\ \beta'(n, q) = 0$
11: $\beta'(N, Q) \leftarrow 1$
12: for $n \leftarrow N \dots 2$ do
13:     for $a \in \text{pre}(n)$ do
14:         $\alpha'_{arc}(0) \leftarrow \alpha'(\text{s}(a), 0) + l(w(a), \epsilon) + \delta$
15:         for $q \leftarrow 1 \dots Q$ do
16:             $\alpha'_{arc}(q) \leftarrow \min \left\{ \begin{array}{c} \alpha'(\text{s}(a), q{-}1) + l(w(a), r_q) \\ \alpha'(\text{s}(a), q) + l(w(a), \epsilon) + \delta \\ \alpha'_{arc}(q{-}1) + l(\epsilon, r_q) \end{array} \right\}$
17:             $b_{arc}(q) \leftarrow$ line chosen in min expression above (1, 2 or 3)
18:         end for
19:         $\forall q, \beta'_{arc}(q) = 0$
20:         for $q \leftarrow Q \dots 1$ do
21:             $\beta'_{arc}(q) \leftarrow \beta'_{arc}(q) + \frac{\alpha(\text{s}(a))p(a)}{\alpha(n)} \beta'(n, q)$
22:             switch $(b_{arc}(q)) \left\{ \begin{array}{ll} 1: & \beta'(\text{s}(a), q{-}1) \mathrel{+}= \beta'_{arc}(q) \\ 2: & \beta'(\text{s}(a), q) \mathrel{+}= \beta'_{arc}(q) \\ 3: & \beta'_{arc}(q{-}1) \mathrel{+}= \beta'_{arc}(q) \end{array} \right\}$
23:             switch $(b_{arc}(q)) \left\{ \begin{array}{ll} 1: & \gamma(q, w(a)) \mathrel{+}= \beta'_{arc}(q) \\ 3: & \gamma(q, \epsilon) \mathrel{+}= \beta'_{arc}(q) \end{array} \right\}$
                // Not C syntax; execute only one line of switch statement
24:         end for
25:         $\beta'_{arc}(0) \leftarrow \beta'_{arc}(0) + \frac{\alpha(\text{s}(a))p(a)}{\alpha(n)} \beta'(n, 0)$
26:         $\beta'(\text{s}(a), 0) \mathrel{+}= \beta'_{arc}(0)$ // Handle special case q=0
27:     end for
28: end for
29: $\forall q, \beta'_{arc}(q) = 0$ // Now do start node (imagine dummy arc to "real" start node)

30: for $q \leftarrow Q \dots 1$ do
31:     $\beta'_{arc}(q) \leftarrow \beta'_{arc}(q) + \beta'(1, q)$ // c.f. Line 21
32:     $\beta'_{arc}(q{-}1) \leftarrow \beta'_{arc}(q{-}1) + \beta'_{arc}(q)$ // c.f. Line 22, case 1
33:     $\gamma(q, \epsilon) \mathrel{+}= \beta'_{arc}(q)$ // c.f. Line 23, case 1
34: end for
35: Check that $\sum_s \gamma(q, s) = 1$ for $1 \leq q \leq |R|$
```

Figure 5: Statistics computation for reference optimization

that while optimizing R, we make sure that there is an ϵ symbol between each real word. This means there is a position between each word where we can accumulate statistics, which makes it possible to store statistics for words in the lattice that are inserted relative to the current hypothesis. We use a function normalize-eps(R) which produces a sequence like ($\epsilon\, w_1\, \epsilon\, w_2\, \ldots\, w_n\, \epsilon$) with $w_1 \ldots w_n$ being real words, and remove-eps(R) which removes the ϵ's from the sequence R. The algorithm starts from the MAP estimate and iteratively improves it. On each iteration we are minimizing the auxiliary function:

$$\mathcal{Q}(R; R', \mathcal{L}) \;\; \equiv \;\; \sum_{q=1}^{|R'|} \sum_{s} \gamma_{R'}(q,s) l(r_q, s) \qquad (13)$$

where we use $\gamma_{R'}(q,s)$ to represent the statistics $\gamma(q,s)$ accumulated from the lattice given the previous sequence R'. The full justification for Equation 13 is quite complicated and is presented in Appendix B. The intuition is that $\gamma_{R'}(q,s)$ represents the probability mass of symbol s that aligned to position q in the reference (given the previous reference value R'), and we give this weight to a corresponding term in the Levenshtein edit-distance expression, $l(r_q, s)$, where r_q is the q'th position in the "new" reference R which we treat as a variable. We compute the value of R that minimizes (13) by setting r_q to the symbol (or ϵ) s with the largest value of $\gamma_{R'}(q,s)$. The way the algorithm handles insertions and deletions is by, respectively, turning a real symbol into ϵ or ϵ into a real symbol. The latter case can only happen if the reference sequence contains epsilons; this is the reason for the normalize-eps step. The need for the small constant δ relates to symbols needing to be inserted into the reference: the constant encourages these extra symbols in the lattice not aligned to real words in R, to align to these ϵ positions rather than aligning to the "gaps" between the positions in R which can result in the corresponding statistics being lost[2] On each iteration we compute an auxiliary function change $\Delta\mathcal{Q}$, and from that iteration to the next the edit distance L will decrease by at least as much as $\Delta\mathcal{Q}$ (this should be checked). Note that we do not prove that our algorithm reaches a local optimum in any sense, only

[2]We have described this quite informally; more precisely, the δ is to avoid taking the middle line of the min expression of line 16 of Figure 5 when an alternative alignment with equivalent edit-distance is available, since this middle line results in the statistics getting "lost" (c.f. line 23).

```
$R$ = MBR-Decode($\mathcal{L}$):
 1: $R \leftarrow$ One-best path of $\mathcal{L}$
 2: loop
 3:     $R \leftarrow$ normalize-eps($R$)
 4:     $(\hat{L}, \gamma(\cdot,\cdot)) \leftarrow$ Acc-Stats($\mathcal{L}, R$)  // Figure 5
 5:     $\Delta\mathcal{Q} \leftarrow 0$
 6:     for $q \leftarrow 1 \ldots |R|$ do
 7:         $\hat{r} \leftarrow \max_s \gamma(q, s)$
 8:         $\Delta\mathcal{Q} \leftarrow \Delta\mathcal{Q} + \gamma(q, r_q) - \gamma(q, \hat{r})$
 9:         $r_q \leftarrow \hat{r}$
10:     end for
11:     if $\Delta\mathcal{Q} = 0$ then
12:         break
13:     end if
14: end loop
15: $R \leftarrow$ remove-eps($R$)
```

Figure 6: Minimum Bayes Risk Decoding Algorithm

that it will never make things worse, so there is no inconsistency between the need to add ϵ symbols and the δ constant to improve optimization, and our proofs which do not rely on the presence of these ϵ's or the δ constant. The proof applies even with ϵ symbols in the sequence and the lattice, because the edit distance as defined in Section 4 is invariant to them.

7.3. Stochastic version

In order to investigate the convergence properties of our algorithm, we also experiment with a version of the algorithm inspired by simulated annealing [11]. At each stage, we choose the symbol s at position q with probability proportional to $\exp(\gamma(q, s)/T)$, where the "temperature" T controls the randomness of this assignment. We assign zero probability where $\gamma(q, s) = 0$. We start the temperature T at some specified value T_{init} and decrease it in intervals of T_{step} until it is zero. We remember the best objective function value we encountered during the optimization, and return it; this is a standard optimization in such approaches. When doing this, we make use of the fact that that the auxiliary function change is an upper bound on the actual objective function change[3], and replace the best objective function value with our best bound on the objective function value after applying one "standard" update. We emphasize that we used this approach primarily as a means to investigate the performance of our main algorithm.

7.4. System combination

It is possible to run our algorithm in system combination mode. When combining multiple lattices generated for the same utterance from multiple systems, in place of the Bayes Risk of Equation (3) we are minimizing a quantity averaged over multiple systems. Suppose we are combining N systems. We first assign a weight λ_i to each system with $i = 1, \ldots, N$, where $\sum_i \lambda_i = 1$ (in our experiments we just used equal weights $\lambda_i = 1/N$). Instead of (3) we aim to minimize:

$$\mathcal{R}(W) = \sum_{i=1}^{N} \lambda_i \sum_{W'} P_i(W'|\mathcal{X}) L(W, W'). \tag{14}$$

Algorithmically, this leads to a very simple change: we compute statistics $\gamma_i(q, s)$ for the systems $1 \leq i \leq N$, and simply average them with:

$$\gamma(q, s) = \sum_{i} \lambda_i \gamma_i(q, s). \tag{15}$$

Then we treat the summed statistics $\gamma(q, s)$ in just the same way as we would if they were obtained from a single lattice. This is in contrast with CNC [4] in which the extra work required to do system combination is nontrivial. Note that it is not possible to simply combine the different lattices into a single lattice, since the overall likelihood of the utterance will in general be different for the different systems, and this will affect the weight of the posteriors in the individual lattices.

In system combination mode, at the beginning of the algorithm we initialize R to the one-best (MAP) output of one of the systems, e.g. the best single system as measured on a development set.

The reason why the MBR decoding framework is useful for system combination is that if we write down the equivalent of (14) for the standard, sentence-level error rate we get a rather trivial result where the output sentence must be present in at least one of the source systems.

[3]I.e. the objective function change may be more negative than the auxiliary function change.

8. Experimental setup

We tested on three different experimental setups: English broadcast news (HUB4), Mandarin dictation and Arabic broadcast news.

8.1. English and Mandarin systems– common features

Our English and Mandarin systems were standard HTK systems [26] with 39 dimensional features (both MFCC and PLP [9]). To have a variety of systems to test on and to combine, we built systems on MFCC and on PLP features, and with either Maximum Likelihood (ML) training, or Minimum Phone Error (MPE) [18], or Boosted Maximum Mutual Information (BMMI) [17]. Decoding and lattice generation were run with a language model scale of 12.0 and a word insertion probability of $\exp(-10.0)$.

Given these scaling factors, the lattice arc likelihoods $p(a)$ which we used in our algorithm were computed as follows, if we write the acoustic log likelihood as $m(a)$ and the language model log probability as $l(a)$:

$$\log p(a) = \kappa(m(a) + 12l(a) - 10), \tag{16}$$

where κ is a scaling factor (as in MPE training [18]) which is necessary to map back to "real" probabilities. We tried two values of κ: the inverse of the language model scale (i.e. 1/12) which is the "natural" value to map back to real probabilities, and 1/15 which had been optimized for MPE training. We found that the former was better for Consensus (the baseline method) and the latter was better for our method, and we present results only with those optimized values. Note that Equation (16) (with the appropriate scale and insertion-penalty values) is the only way to maintain consistency with the decoder, and the scale κ is standard in lattice processing algorithms (although in some cases it is fixed at the inverse of the language model scale).

For all experiments we used $\delta = 10^{-4}$. We did not tune this because we do not expect the algorithm to be sensitive to it as long as it is small and positive. We implemented all our algorithms in double precision arithmetic, since these types of algorithms can be very sensitive to numerical round-off.

For the English and Mandarin tasks, the decoding and system combination methods we compared were as follows: MAP decoding (the baseline), for which we just used the 1-best output of the decoder; Consensus, for which we used the SRILM toolkit; and our own method, which we implemented using `C` code in the HTK framework. For system combination, we used our own implementation of ROVER since it performed slightly better than the SRILM version. For Confusion Network Combination (CNC) we used the SRILM toolkit; the algorithm implemented in SRILM for creating "word meshes" and combining them using `nbest-lattice` is a slight variant of CNC. Our own algorithm was run in system combination mode by combining the statistics from the lattices on each iteration of the algorithm as in Equation (15), with equal weights for each lattice. Rather than tuning the selection of systems for system combination we just picked them in order from the best to the worst, based on the WER of the individual systems. The pruning beams we used were 25 for Consensus and 80 for our method (before multiplying by κ); after scaling by the scales we used for those methods these become 2.08 and 5.33 respectively.

8.2. English broadcast news

Our English broadcast news system was trained on 144 hours of data, consisting of the HUB4-97 training set (74 hours) and the HUB4-98 training set (70 hours). We tested on the NIST BN 1999 test data, which is about 2.5 hours long with 735 segments. Our models had 5.9k shared states and 16 Gaussians per state. We used a lexicon with 60k words; trigram language models were trained using the HTK tools on the text data provided for the HUB4 task.

As an example of the lattice characteristics for the English system, our MLE system with MFCC features had lattices with an average 85 arcs per reference word, and an oracle word error rate of 13.4% (versus a WER of 27.6%). The corresponding confusion networks had an oracle error rate of 18.6% and on average 1.92 arcs per confusion network bin.

8.3. Mandarin dictation

The training data for our Mandarin dictation task consists of 113 hours of Chinese 863-project data [19, 27] and 31.5 hours of data collected by Microsoft Research Asia (MSRA) [2], giving 144.5 hours overall. We tested on a database containing 1.52 hours of data, also from MSRA [22]. This contains 1000 utterances from 50 test subjects; these were not further segmented. Our language model training data consists of about 60MB of People's Daily text from the first half of 1998, plus about 30MB of data collected from various Chinese websites. We used HTK tools to train a trigram language model. Our models had 6.2k shared states and 16 Gaussians per state.

In order to make our scoring insensitive to word segmentation issues which arise in Mandarin, error rate evaluation was done at a character sequence level, i.e. we measure Character Error Rate (CER). To make our MBR decoding consistent with this metric, we first expanded our lattices from words to characters and ran our algorithm at the character level.

As an example of our lattice characteristics, for the ML estimated MFCC system our lattices had an oracle CER of 5.3%, compared to a CER of 18.4% for the baseline decoding. The average number of arcs per reference word was 62. The corresponding confusion networks had an oracle error rate of 7.3% and on average 2.36 arcs per confusion network bin.

8.4. Arabic broadcast news

Experiments on Arabic broadcast news were done at IBM using a different experimental setup from the one for English and Mandarin. The system used was built for DARPAs Global Autonomous Language Exploitation (GALE) project. In [20], the previous year's version of the system is described. The differences between that system and the one we used here are probably not important for our purposes. We do combination experiments on three different decoding outputs, which we call `UxSGMM`, `SGMMxU`, and `V_NNxSGMMxU`. The naming convention is that `XxY` means system `X`, adapted using as supervision the transcripts from system `Y`. The `SGMM` system is a "Subspace Gaussian Mixture Model" as described in [16, 20, 3]. This system is "vowelized", i.e. the lexicon contains Arabic short vowels. The `U` (unvowelized) system lacks short vowels. The `V_NN` system is the combination of a vowelized system (`V`) and a system using neural network features [16] (`NN`), built using separate context-dependency trees but decoded as a single model by combining log likelihoods. The likelihoods $p(a)$ on arcs in the lattice are products of the language model probability with scaled-down acoustic probabilities; the acoustic scales are system dependent, typically 1/12 for SGMM systems and 1/18 for conventional systems. The IBM system does not use an insertion penalty. All results are given on the Dev'09 test set, which was released by the Linguistic Data Consortium for the GALE Phase 4 evaluation. It is three hours long. Our segmentation of the test set had 1290 segments, with 16 words on average per segment. The lattice depth measured for the `V_NN` system was 119 arcs per reference word on average. On average for the three systems, the confusion network depth was 2.3% and the confusion network oracle error rate was 6.5% (about half the %WER, c.f. Table 3).

Our algorithm was re-implemented in a different language (`perl`) to accept the Finite State Transducer based lattice format used for IBM's system; we verified that this implementation gave the same results as our `C` code when reading HTK lattices. The SRILM toolkit was used for N-best ROVER system combination, but local implementations were used for Consensus and CNC. We did not prune lattices, in our algorithm; for Consensus, we pruned away links with posterior less than 10^{-3} (after scaling acoustic probabilities down by the inverse language model scale). All results were scored with NIST's `sclite` tool, which requires timing information; this necessitated an extension to our algorithm to store time information, which is described in Appendix C.

9. Experimental results

9.1. Single lattice decoding

Decoding	MFCC			PLP		
Method	MLE	BMMI	MPE	MLE	BMMI	MPE
MAP	27.55	25.59	24.68	27.83	25.67	24.69
Consensus	27.30	25.34	24.37	27.47	25.32	24.45
Proposed	**27.19**	**25.12**	**24.28**	**27.19**	**25.14**	**24.19**

Table 1: English BN, single system: %WER

Decoding	MFCC			PLP		
Method	MLE	BMMI	MPE	MLE	BMMI	MPE
MAP	17.95	17.36	17.15	18.39	17.71	17.70
Consensus	17.79	17.18	**16.74**	18.20	17.59	17.49
Proposed	**17.70**	**17.09**	16.76	**18.03**	**17.43**	**17.48**
(Proposed, word-level)	18.11	17.65	17.50	18.65	18.06	17.95

Table 2: Mandarin, single system: %CER

Table 1 shows results on English broadcast news. For six different systems, the ordering of results is consistent: Consensus is better than MAP decoding, and our method is better than Consensus. Table 2 shows similar results on Mandarin, which are almost as unamimous (in one case, Consensus was very slightly better). Note the extra row "Proposed, word-level". In this experiment, we ran our technique using the original lattices, rather than lattices expanded into characters: that is, we optimized for WER while

Decoding	System		
Method	`UxSGMM`	`V_NNxSGMM`	`SGMMxU`
MAP	12.9	13.0	13.1
Consensus	12.9	13.0	13.1
Proposed	12.9	13.2	13.2

Table 3: Arabic (IBM systems): %WER (Dev'09)

testing CER. The results were worse than the baseline MAP decoding. This indicates that MBR decoding is very targeted to the exact metric being optimized.

Table 3 shows results on Arabic. This is for a very different setup, based at IBM, with different tools and a different implementation of our algorithm. These results are less encouraging. Consensus gives no improvement, and our method slightly increases the WER. Note that we are not too surprised that Consensus gave no improvement, because in our experience it does not typically help at very low Word Error Rates (in this region, the sentence and word error rates are more correlated and there is less difference to exploit). As to why our technique degrades performance slightly, this may be due to differences in the lattice generation or in the models. Unlike our experiments on HTK-based systems, for our experiments at IBM we did not map silence and noise words and sentence start and end tokens to ϵ. Not doing so was found to be slightly better (0.1% on one setup). For Consensus this made no difference to WER.

9.2. System combination

Combination Method	2-way	3-way	4-way
Pick best (MAP)	24.68	24.68	24.68
MAP+ROVER	24.24	23.81	23.85
10Best Rover	23.62	23.62	23.77
CNC	23.31	23.31	22.95
Proposed	**23.05**	**23.05**	**22.82**

Table 4: English BN, system combination: %WER

Tables 4 and 5 show system combination results for our English and Mandarin systems. We ran this for 2, 3 and 4-system combination; we just combined the N systems with the best WERs in each case. We do not show

Combination Method	2-way	3-way	4-way
Pick Best (MAP)	17.15	17.15	17.15
MAP+ROVER	17.09	16.33	16.36
10Best ROVER	16.02	15.96	15.86
CNC	15.82	15.73	15.69
Proposed	**15.72**	**15.62**	**15.57**
(Proposed, word-level)	16.68	16.55	16.55

Table 5: Mandarin, system combination: %CER

Combination Method	3-way
Pick Best (MAP)	12.9
100Best ROVER	12.3
Consensus	12.2
Proposed	12.2

Table 6: Arabic, system combination: %WER (Dev'09)

results for more than 4 systems because results did not improve further. We combine systems in order from best to worst (this makes a difference in ROVER and CNC, and in our method it makes a difference because of initialization to the MAP output of the first system). The baseline "Pick best (MAP)" refers to picking the best single system (over the whole test set), decoded with MAP. The baselines are MAP+ROVER which is normal (MAP) decoding plus ROVER; 10-best sentence decoding from the lattices followed by ROVER; Confusion Network Combination (CNC); and our proposed method. In all cases our method is the best. For the Mandarin system we also ran our method in a mismatched manner to optimize WER rather than CER, and as before this degrades performance, but combination still helps even with the mismatch.

Table 6 shows results on IBM's Arabic language systems, in combination model. Here, our method gives the same improvement as Consensus. All methods give about the same improvement here.

9.3. Speed

We measured the speed of our method; this was done for single systems. In system combination mode the time taken scales proportionally to the number of systems being combined. In the HTK-based systems with a `C` implementation, our method was about 100 times faster than real time; for comparison, Consensus on this setup was about 200 times faster than real time[4]. For the IBM systems, our method implemented in `perl` was two times slower than real time[5]. We believe much of the difference between the setups is due to the overhead of an interpreted language or other aspects of the implementation, since the IBM lattices are not significantly deeper than the others and the segments are slightly shorter. The same `perl`-based software applied to HTK lattices gave a speed of two times faster than real time. The pruning beam of 80 we use with the HTK setup is very wide and has a negligible effect on speed (versus un-pruned) so pruning is not an issue.

If the number of arcs in a lattice is N and the average length of word-sequences in the lattice is L, then Consensus is $O(N^3)$ and ours is $O(NL)$ (and bear in mind that $L \leq N$). However, Consensus works best for highly pruned lattices and our method is designed to work well without pruning, so the constant factor in the speed is more favorable to Consensus[6]. As mentioned in Section 8, the pruning beams were much tighter for Consensus than our method. We believe that in most cases, when implemented in a compiled language, compute time should not be a limiting factor in applying this method.

9.4. Lattice expansion experiments: investigating tightness of bound

We performed lattice expansion experiments in order to test the level of approximation in our algorithm (Figure 4) for computing the edit distance between a sequence and a lattice. We prove in Appendix A that it computes an upper bound on the true edit distance; here we investigate how close that bound is in practice. The basic idea is this: if we were to expand out the paths in the lattices so that they all ran in parallel, with no shared arcs, then it is easy to see that our algorithm would be exact (for $\delta = 0$). This is impractical but we can expand the lattice to varying degrees so that the most likely paths share as few arcs as possible. The limit of such an expansion process is the situation where our algorithm is exact, so we try to see whether

[4]All speeds on HTK lattices are measured on an AMD Athlon 64 bit dual core processor (using one core), running at 2.2GHz

[5]On an Intel Xeon CPU running at 2.8GHz

[6]For these $O(N)$ results to hold formally for our technique, we need to specify a maximum number of iterations in Algorithm 6, e.g. force a stop after 10 iterations regardless of convergence. In practice it always terminates after about one to four iterations so we do not bother with this.

our measured lattice edit distance appears to be converging to something as we expand more of the lattice. We used a lattice expansion algorithm that identifies arcs with probability $\gamma(a)$ greater than some threshold (e.g. 0.01), and if they end in a node with multiple incoming nodes, the algorithm splits that node into two copies, one with a transition from the identified arc and one with a transition from all the other arcs. This ensures more unique left context for the following arcs. We then varied this threshold and measured the average of the approximated edit distance $\hat{L}(\mathcal{L}, R)$ where R is our original (MAP) hypothesis, over the English BN test set. In Figure 7 this is plotted against lattice depth, which varies with the threshold used in the expansion algorithm. As expected the edit distance decreases monotonically with increasing expansion of the lattices, but the differences are tiny. The largest difference in average $\hat{L}$ value is from 4.76991 with our original lattices, to 4.76973 with a lattice depth of 189 (this is with a threshold of 0.002). The average $\hat{L}$ value after optimizing the reference is 4.56041, so the change due to expansion is a tiny 0.09% of the change due to optimization. Since the lattice expansion algorithm is not symmetric in time we also experimented with a "reflected" version; this did not change the MBR criterion at all (this is due to interaction with time-asymmetry in our edit-distance algorithm[7]).

Our conclusion is that for all practical purposes, our lattice edit distance computation (given in Figure 4) can be considered exact for speech data. Most of the inaccuracy in our method comes from the optimization procedure (see below).

9.5. Stochastic method: testing effectiveness of optimization

In order to gain insight into the convergence properties of our procedure for optimizing the sequence R, we carried out experiments with the stochastic update method which we described in Section 7.3. We believe it should be possible to show that with sufficiently many iterations, the stochastic approach would give us, with probability approaching 1, a perfect global minimum of the approximated lattice edit distance computed by the algorithm of Figure 4.

Our experiments used the English BN system, with an MPE-trained model that had 12 Gaussians per state. Figure 8 shows the optimization

[7]We can formalize the two expansion algorithms as approaching, respectively, a right-branching tree and a left-branching tree. We can then show that our algorithm is exact for a right-branching tree so the limit of the first algorithm corresponds to exact edit distance.

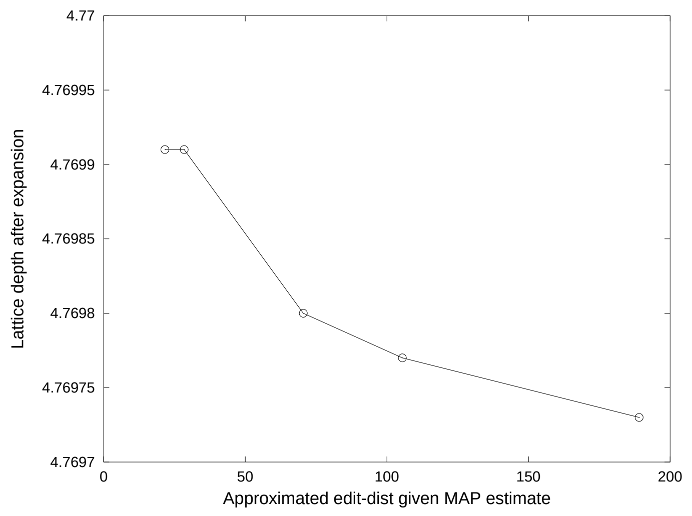

Figure 7: Lattice expansion– effect on edit distance

performance of our stochastic update method for different numbers of iterations. We plot this for various T_{init} values; the higher the starting temperature, the more "randomized" the algorithm. The x axis is $T_{\text{init}}/T_{\text{step}}$ which is the approximate number of iterations. On the y axis we show the objective function, which equals the average over all test files, of the of the approximated edit distance $\hat{L}$ computed in the stochastic algorithm. The scale of the y axis is related to the average length of utterances so the absolute values are not meaningful. To avoid compressing the scale of the graph we do not display the objective function value prior to optimizing the sequence, which was 5.244.

The best objective function value on this test set, obtained with $T_{\text{init}} = 0.2$ after 200 iterations, is 4.995. Compare with the objective function measured at the MAP estimate, at 5.244, and with our basic, deterministic algorithm which gives 5.014. Our deterministic algorithm gives 7.6% less objective function improvement over the MAP baseline than the best figure we obtained using the stochastic approach. This percentage (7.6%) can be taken as a lower bound on how much relative gain we are losing from a purely optimization point of view: i.e. once we decide that we will minimize the "approximate lattice edit distance" as computed by the algorithm in Figure 4, how well is the algorithm of Figure 6 minimizing that? The WER improvement from our method is only 0.5% absolute so the most we could hope to gain from our stochastic improvement is about 7.6% of this, or 0.04% absolute. This cannot reliably be detected without a very much larger test set than ours, so we omit WER results for this experiment; these showed no significant differences. The conclusion we draw is that the basic algorithm we presented is optimizing the objective function sufficiently well, and very little could be gained from further work on it.

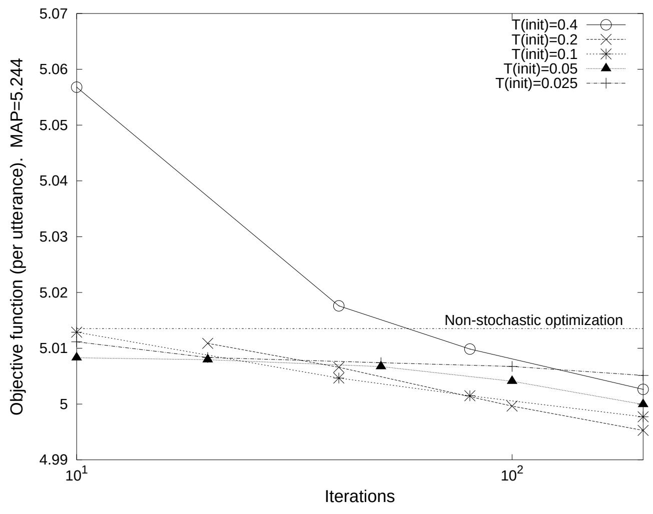

Figure 8: Stochastic vs. deterministic optimization

10. Conclusions

We have described an algorithm that computes an approximation to the weighted edit distance between a word sequence and a lattice, and an algorithm that optimizes the word sequence to minimize that approximated value. We have applied this to Minimum Bayes Risk decoding problem for speech recognition. It has previously been established that optimizing the Bayes Risk with respect to the word-level, rather than sentence-level, error can improve Word Error Rates. Our work here confirms this, and our method leads to consistently better Word Error Rate improvements than Consensus on the HTK-based systems we tested. However, for a different setup at IBM we did not see improvements. We hope that future experiments will establish which setup is more typical.

We emphasize that our method is conceptually cleaner than Consensus and has broader applicability (since it does not rely on phonetic similarity or time information), so even if the WER results are the same, our method still has value. Whereas most previous algorithms for MBR decoding are heuristic in nature, our algorithm can be shown to iteratively reduce an upper bound on the edit-distance. We do lattice expansion experiments that hint that our upper bound on the edit distance is extremely close to optimal for speech recognition lattices, and experiments with randomized updates indicate that our optimization method is probably about 90% effective (i.e. we get about 90% of the possible objective function improvement). Thus we are confident that for all practical purposes we have exactly optimized the Bayes Risk metric.

Appendix A. Bound for Approximated edit distance

Here we prove that the approximated edit distance $\hat{L}(\mathcal{L}, R)$ is an upper bound on the true edit distance $L(\mathcal{L}, R)$. For simplicity we prove the bound for zero δ, but it is trivial to extend it to $\delta > 0$. In Appendix B we will prove that our algorithm optimizes our approximated edit distance (for any δ).

The approximated edit distance $\hat{L}(\mathcal{L}, R)$ is computed via the following recursion, which is the same as that used in Algorithm 5. Below, $\alpha'(a, q)$, which is defined for arcs a and $0 \leq q \leq |R|$, is the "forward edit distance" computed at arcs.

$$\alpha'_R(a, q) \;:=\; \min \left\{ \begin{array}{c} \alpha'_R(\mathrm{s}(a), q{-}1) + l(w(a), r_q) \\ \alpha'_R(\mathrm{s}(a), q) + l(w(a), \epsilon) + \delta \\ \alpha'_R(a, q{-}1) + l(\epsilon, r_q) \end{array} \right\} \tag{A.1}$$

This refers to a quantity $\alpha'_R(n, q)$, defined for nodes n and $0 \leq q \leq |R|$:

$$\alpha'_R(n, q) \;:=\; \frac{\sum_{a \in \mathrm{pre}(n)} \alpha(a) \alpha'_R(a, q)}{\alpha(n)}, n \neq b(\mathcal{L}) \tag{A.2}$$

$$\alpha'_R(b(\mathcal{L}), 0) \;:=\; 0 \tag{A.3}$$

$$\alpha'_R(b(\mathcal{L}), q) \;:=\; \alpha'_R(b(\mathcal{L}), q{-}1) + l(\epsilon, r_q), \;\; q > 0. \tag{A.4}$$

To simplify the equations we assume that any options in these min expression referring to out-of-bounds quantities ($q = -1$) are not taken (equivalently we can define $\alpha'(x, -1) = +\infty$). The edit distance is given by:

$$\hat{L}(\mathcal{L}, R) := \alpha'_R(\mathrm{e}(\mathcal{L}), |R|). \tag{A.5}$$

It is easy to verify that this is the same as the $\hat{L}$ quantity computed in the algorithm of Figure 4.

Recall that paths are sequences of arcs (Sec. 6). We use the notation $\{S : S \in a\}$ for the set of paths starting at the start node and ending in arc a, and $\{S : S \in n\}$ for the set of paths starting at the start node and ending in an arc whose end node is n. This notation is a short-hand since nodes and arcs are not really sets of paths. The set $\{S : S \in b(\mathcal{L})\}$ is the set containing one empty path $\emptyset$ which has likelihood $p(\emptyset) = 1$. We assume all path likelihoods are nonzero, so all α quantities are positive. We will sometimes use a single variable x that can refer to either arcs or nodes, so $S \in x$ would be one of the above definitions depending whether x is an arc or a node.

We make use of the α quantities used in the standard forward-backward algorithm:

$$\begin{aligned} \alpha(n) &:= \sum_{a \in \mathrm{pre}(n)} \alpha(a),\ n \neq b(\mathcal{L}) && \text{(A.6)} \\ \alpha(a) &:= \alpha(\mathrm{s}(a))p(a) && \text{(A.7)} \\ \alpha(b(\mathcal{L})) &:= 1 && \text{(A.8)} \end{aligned}$$

The α quantities are sums of path likelihoods up to the current point:

$$\alpha(x) = \sum_{S \in x} p(S) \qquad \text{(A.9)}$$

This is a standard result used in proofs of the Forward-Backward algorithm.

We now introduce the concept of an *alignment* $\mathcal{A}$ of a lattice to a word sequence. An alignment tells us what point in the word sequence the beginning of an arc aligns to, given the ending position of the arc. It may be thought of as a set of back pointers. Formally, an alignment $\mathcal{A}$ is a function from an arc a and arc-end position $0 \leq q \leq |R|$, to an arc-start position $0 \leq q' \leq q$, i.e.

$$q' = \mathcal{A}(a, q). \qquad \text{(A.10)}$$

We define a path edit distance function $L(S, R, \mathcal{A})$ *given an alignment* as:

$$\begin{aligned} L(S, R, \mathcal{A}) &:= L(W(S_1^{|S|-1}), R_1^{\mathcal{A}(s_{|S|}, |R|)}, \mathcal{A}) && \text{(A.11)} \\ &\quad + L((w(s_{|S|})), R_{\mathcal{A}(s_{|S|}, |R|)+1}^{|R|}),\ S \neq \emptyset \end{aligned}$$

where we use $s_{|S|}$ to extract the last element of S. The base case for the empty sequence $\emptyset$ as:

$$L(\emptyset, R, \mathcal{A}) \ := \ L(\emptyset, R). \tag{A.12}$$

It is clear that

$$L(S, R, \mathcal{A}) \geq L(W(S), R), \tag{A.13}$$

since the edit distance on word sequences can be defined as a minimum over all alignments. We will take this as given. We next define the edit distance from a lattice to a word-sequence, given an alignment, as:

$$L(\mathcal{L}, R, \mathcal{A}) \ = \ \frac{\sum_{S \in \mathcal{L}} p(S) L(S, R, \mathcal{A})}{\sum_{S \in \mathcal{L}} p(S)}. \tag{A.14}$$

Theorem 1. *If* $\delta = 0$ *(cf.* (A.1)*), then:*

$$\hat{L}(\mathcal{L}, R) = \min_{\mathcal{A}} L(\mathcal{L}, R, \mathcal{A}). \tag{A.15}$$

We will prove this below. This immediately leads to:

Corollary 1. *If* $\delta = 0$*, then our computed edit distance is an upper bound on the exact one:*

$$\hat{L}(\mathcal{L}, R) \geq L(\mathcal{L}, R). \tag{A.16}$$

Corollary 1 follows from (A.13), (A.14), (A.15) and (12).

We will use $\hat{\mathcal{A}}$ to denote the alignment $\mathcal{A}$ that is implicitly defined by the recursions of (A.1) to (A.4). This $\hat{\mathcal{A}}$ is actually a function of $\mathcal{L}$ and R, but we do not make this explicit because it would clutter the notation. We define the choose-min operator such that choose-min$\left\{ \begin{array}{c} a_1 \rightarrow b_1 \\ a_2 \rightarrow b_2 \end{array} \right\}$ evaluates to b_1 if $a_1 \leq a_2$, and otherwise to b_2. In general, it returns the b_n corresponding to the lowest a_n, choosing, say, the lowest-numbered option in the case of a tie. We define (cf. (A.1)),

$$\hat{\mathcal{A}}(a, q) \ := \tag{A.17}$$
$$\text{choose-min} \left\{ \begin{array}{rcl} \alpha'_R(\mathrm{s}(a), q{-}1) + l(w(a), r_q) & \rightarrow & q{-}1 \\ \alpha'_R(\mathrm{s}(a), q) + l(w(a), \epsilon) + \delta & \rightarrow & q \\ \alpha'_R(a, q{-}1) + l(\epsilon, r_q) & \rightarrow & \mathcal{A}(a, q{-}1) \end{array} \right\}.$$

Lemma 1. *The α' quantities are upper bounds on a weighted edit distance up to the current point, given the alignment $\hat{\mathcal{A}}$ defined by the recursion. This is actually true with equality, but at this point we only establish the bound. Formally, for $0 \leq q \leq |R|$, with x an arc or a node,*

$$\alpha'_R(x, q) \quad \geq \quad \frac{\sum_{S \in x} p(S) L(S, R_1^q, \hat{\mathcal{A}})}{\sum_{S \in x} p(S)}\,. \tag{A.18}$$

Proof of Lemma 1. We use an inductive argument based on a partial ordering on the pairs (x, q) where x is an arc or a node. Using the obvious partial order on the arcs and nodes in the graph where the start node is first, we define the following partial order on pairs:

$$(x, q) \leq (x', q') \text{ iff } x \leq x' \wedge q \leq q'. \tag{A.19}$$

If (A.18) is violated for any pair, then we can choose some (not necessarily unique) pair which violates (A.18) and which is "earliest" in the sense that no other such pair precedes it in the partial ordering. Let the chosen pair be (x, q). We will consider three distinct cases, where x represents *A)* a non-initial node, *B)* an arc, and *C)* the start node.

A) x is a non-initial node $n = x$. We assume (A.18) is false:

$$\alpha'_R(n, q) < \frac{\sum_{S \in n} p(S) L(S, R_1^q, \hat{\mathcal{A}})}{\sum_{S \in n} p(S)}. \tag{A.20}$$

The set of paths $S \in n$ is the same as the union of the sets of paths $S \in a$ for $a \in \text{pre}(n)$; these sets are disjoint. Rewriting (A.20) as a sum over arcs, and using definition (A.2):

$$\frac{\sum_{a \in \text{pre}(n)} \alpha(a) \alpha'_R(a, q)}{\alpha(n)} < \frac{\sum_{a \in \text{pre}(n)} \sum_{S \in a} p(S) L(S, R_1^q, \hat{\mathcal{A}})}{\sum_{S \in n} p(S)} \tag{A.21}$$

The denominators on the left and right are the same by (A.9), and positive. Canceling them, and observing that if (A.21) is true for the sum over $a \in \text{pre}(n)$ then it must be true for some particular a, we have for some $a \in \text{pre}(n)$,

$$\alpha(a) \alpha'_R(a, q) < \sum_{S \in a} p(S) L(S, R_1^q, \hat{\mathcal{A}}). \tag{A.22}$$

Dividing by (A.9) , this is (A.18) except with $<$ not $\geq$, which is a contradiction because $(a, q) < (x, q)$ under the ordering.

B) Now suppose the "earliest" element is an arc a, i.e.

$$\alpha'_R(a,q) < \frac{\sum_{S \in a} p(S) L(S, R_1^q, \hat{\mathcal{A}})}{\sum_{S \in a} p(S)}. \tag{A.23}$$

We consider each of three alternatives depending on the value of $\hat{\mathcal{A}}(a, q)$.

i) One symbol consumed, $\hat{\mathcal{A}}(a,q) = q{-}1$. This corresponds to the top line of (A.1) and (A.17). From (A.1) (top line),

$$\alpha'_R(a,q) = \alpha'_R(\mathrm{s}(a), q{-}1) + l(w(a), r_q). \tag{A.24}$$

We use the fact that the set of paths $S \in a$ is the same as the set of paths $S \in \mathrm{s}(a)$, with each path extended by a. Starting from (A.23), using (A.11) to expand $L(S, R_1^q, \hat{\mathcal{A}})$, and applying $L((p),(q)) = l(p,q)$, we get:

$$\alpha'_R(a,q) < \frac{\sum_{S \in \mathrm{s}(a)} p(S) L(S, R_1^{q-1}, \hat{\mathcal{A}})}{\sum_{S \in \mathrm{s}(a)} p(S)} + l(w(a), r_q). \tag{A.25}$$

Substituting (A.24) for $\alpha'_R(a,q)$, and canceling $l(w(a), r_q)$,

$$\alpha'_R(\mathrm{s}(a), q{-}1) < \frac{\sum_{S \in \mathrm{s}(a)} p(S) L(S, R_1^{q-1}, \hat{\mathcal{A}})}{\sum_{S \in \mathrm{s}(a)} p(S)}, \tag{A.26}$$

which is a violation of (A.32) for an earlier pair $(\mathrm{s}(a), q{-}1)$ and thus a contradiction.

ii) Zero symbols consumed, $\hat{\mathcal{A}}(a,q) = q$. This is the middle line of (A.1) and (A.17). The argument here is very similar to the argument for the top line (above), and we will not write it down.

iii) Multiple symbols consumed[8]*, $\hat{\mathcal{A}}(a,q) = \hat{\mathcal{A}}(a, q{-}1)$.* This is the bottom line of (A.1) and (A.17). From (A.1) (bottom line),

$$\alpha'_R(a,q) = \alpha'_R(a, q{-}1) + l(\epsilon, r_q). \tag{A.27}$$

Substituting (A.27) into the l.h.s. of (A.23), and (A.11) into the r.h.s.,

$$\begin{pmatrix} \alpha'_R(a, q{-}1) \\ +l(\epsilon, r_q) \end{pmatrix} < \frac{\sum_{S \in a} p(S) \Big(L(\ldots) + L((w(a)), R^q_{\hat{\mathcal{A}}(a,q-1)+1}) \Big)}{\sum_{S \in a} p(S)} \tag{A.28}$$

[8]The text "multiple symbols consumed" is for intuition only

with the shorthand $L(\ldots) = L(S_1^{|S|-1}, R_1^{\hat{\mathcal{A}}(a,q-1)}, \hat{\mathcal{A}})$. We can now use the inequality

$$L((w(a)), R^q_{\hat{\mathcal{A}}(a,q-1)+1}) \leq L((w(a)), R^{q-1}_{\hat{\mathcal{A}}(a,q-1)+1}) + l(\epsilon, r_q) \tag{A.29}$$

which derives from the Levenshtein recursion of (7) since it corresponds to one of the expressions in the min, to obtain:

$$\alpha'_R(a, q-1) < \frac{\sum_{S\in a} p(S) \left(L(\ldots) + L(w(a), R^{q-1}_{\hat{\mathcal{A}}(a,q-1)+1}) \right)}{\sum_{S\in a} p(S)}, \tag{A.30}$$

and we can transform the two $L(\ldots)$ expressions back into the form $L((w(a)), R_1^{q-1}, \mathcal{A})$ using (A.11). This is a violation of (A.18) for a lesser pair $(a, q-1)$ which is a contradiction.

C) x is the start node $x = b(\mathcal{L})$. As mentioned, $\{S : S \in b(\mathcal{L})\}$ contains just the empty sequence $\emptyset$, with path likelihood $p(\emptyset) = 1$. Using this and the base case of the recursion of (A.12), if (A.18) is violated then

$$\alpha'_R(b(\mathcal{L}), q) < L(\emptyset, R_1^q). \tag{A.31}$$

It is easy to see from (A.3) and (A.4) that the left and right hand sides are the same, so we have a contradiction. □

Lemma 2. *The α' quantities are lower bounds on the minimum weighted edit distance up to the current point given any possible alignment:*

$$\alpha'_R(x, q) \quad \leq \quad \min_{\mathcal{A}} \frac{\sum_{S\in x} p(S) L(S, R_1^q, \mathcal{A})}{\sum_{S\in x} p(S)}. \tag{A.32}$$

Proof of Lemma 2. We postulate an alignment $\mathcal{A}$ for which (A.32) is violated, and derive a contradiction. For brevity, let us define the quantity $\tilde{\alpha}_R(x, q, \mathcal{A})$ where x may be either a node or an arc, corresponding to the right hand side of (A.32):

$$\tilde{\alpha}_R(x, q, \mathcal{A}) \quad := \quad \frac{\sum_{S\in x} p(S) L(S, R_1^q, \mathcal{A})}{\sum_{S\in x} p(S)}. \tag{A.33}$$

As for the proof of Lemma 1, we consider the "earliest" position (x, q) for which

$$\alpha'_R(x, q) > \tilde{\alpha}_R(x, q, \mathcal{A}), \tag{A.34}$$

i.e. for which (A.32) is false, and derive a contradiction. Again we have three separate cases according whether x is *A)* a non-initial node, *B)* an arc, or *C)* the start node.

A) x is a non-initial node n. We expand the left side of (A.34) with (A.2) and the right hand side by summing over incoming arcs:

$$\frac{\sum_{a\in \mathrm{pre}(n)}\alpha(a)\alpha'_R(a,q)}{\sum_{a\in \mathrm{pre}(n)}\alpha(a)} > \frac{\sum_{a\in \mathrm{pre}(n)}\sum_{S\in a} p(S)L(S,R_1^q,\mathcal{A})}{\sum_{a\in \mathrm{pre}(n)}\sum_{S\in a} p(S)}. \tag{A.35}$$

We use (A.9) to cancel the denominators. We then note that if the inequality holds then for some $a \in \mathrm{pre}(n)$,

$$\alpha(a)\alpha'_R(a,q) > \sum_{S\in a} p(S)L(S,R_1^q,\mathcal{A}). \tag{A.36}$$

Dividing this by (A.9), we arrive at (A.34) for a lesser pair (a,q) which is a contradiction.

B) x is an arc $x = a$. We handle separate cases depending on the value of $\mathcal{A}(a,q)$.

i) No symbol consumed, $\mathcal{A}(a,q) = q$. We start from (A.34) and derive a contradiction. We use the fact that each path $S' \in a$ is the same as a path $S \in \mathrm{s}(a)$, except with one more element in the sequence (i.e., a). We use $p(S') = p(S)p(a)$ by (10); substituting into (A.33) and canceling $p(a)$,

$$\tilde{\alpha}_R(a,q,\mathcal{A}) = \frac{\sum_{S\in \mathrm{s}(a)} p(S)L((S,a),R_1^q,\mathcal{A})}{\sum_{S\in \mathrm{s}(a)} p(S)}. \tag{A.37}$$

Using $\mathcal{A}(a,q) = q$ and (A.11), and $L((w),\emptyset) = l(w,\epsilon)$,

$$\tilde{\alpha}_R(a,q,\mathcal{A}) = \frac{\sum_{S\in \mathrm{s}(a)} p(S)L(S,R_1^q,\mathcal{A}) + l(w(a),\epsilon)}{\sum_{S\in \mathrm{s}(a)} p(S)}. \tag{A.38}$$

This reduces to $\tilde{\alpha}_R(a,q,\mathcal{A}) = \tilde{\alpha}_R(\mathrm{s}(a),q,\mathcal{A}) + l(w(a),\epsilon)$. Substituting this into the r.h.s. of (A.34) and simplifying,

$$\alpha'_R(a,q) > \tilde{\alpha}_R(\mathrm{s}(a),q,\mathcal{A}) + l(w(a),\epsilon). \tag{A.39}$$

But $\alpha'_R(a,q) \leq \alpha'_R(\mathrm{s}(a),q) + l(w(a),\epsilon)$ by (A.1) (recalling that $\delta = 0$ here). So

$$\alpha'_R(\mathrm{s}(a),q)+l(w(a),\epsilon) \geq \alpha'_R(a,q) > \tilde{\alpha}_R(\mathrm{s}(a),q,\mathcal{A})+l(w(a),\epsilon)$$

and subtracting $l(w(a), \epsilon)$ we have $\alpha'_R(\mathrm{s}(a), q) > \tilde{\alpha}_R(\mathrm{s}(a), q, \mathcal{A})$ which is a contradiction.

ii) One symbol consumed, $\mathcal{A}(a, q) = q{-}1$. The steps here are very similar to case *i)* above, and we omit the details.

iii) Multiple symbols consumed, $\mathcal{A}(a, q) < q{-}1$. This case is a little more complicated. Let us define $p := \mathcal{A}(a, q)$ for brevity, so $p < q{-}1$, and it follows from (A.33) and (A.11) that

$$\tilde{\alpha}_R(a, q, \mathcal{A}) = \tilde{\alpha}_R(\mathrm{s}(a), p, \mathcal{A}) + L((w(a)), R^q_{p+1}). \tag{A.40}$$

Now consider the edit distance expression $L((w(a)), R^q_{p+1})$. It is possible to show that for some (not necessarily unique) position o with $p < o \leq q$, this corresponds to an alignment where $w(a)$ corresponds to position o in R, i.e. an alignment $\begin{array}{ccccc} \epsilon & \ldots & w(a) & \ldots & \epsilon \\ r_{p+1} & \ldots & r_o & \ldots & r_q \end{array}$, so

$$L((w(a)), R^q_{p+1}) = \sum_{n=p+1}^{o-1} l(\epsilon, r_n) + l(w(a), r_o) + \sum_{n=o+1}^{q} l(\epsilon, r_n). \tag{A.41}$$

Substituting this into (A.40),

$$\begin{aligned} \tilde{\alpha}_R(a, q, \mathcal{A}) \quad &= \quad \tilde{\alpha}_R(\mathrm{s}(a), p, \mathcal{A}) + \sum_{n=p+1}^{o-1} l(\epsilon, r_n) && \text{(A.42)} \\ &\qquad + l(w(a), r_o) + \sum_{n=o+1}^{q} l(\epsilon, r_n). && \text{(A.43)} \end{aligned}$$

We can show that for any $p < o \leq q$, the following hold:

$$\begin{aligned} \alpha'_R(\mathrm{s}(a), o{-}1) \quad &\leq \quad \alpha'_R(\mathrm{s}(a), p) + \sum_{n=p+1}^{o-1} l(\epsilon, r_n) && \text{(A.44)} \\ \alpha'_R(a, o) \quad &\leq \quad \alpha'_R(\mathrm{s}(a), o{-}1) + l(w(a), o) && \text{(A.45)} \\ \alpha'_R(a, q) \quad &\leq \quad \alpha'_R(a, o) + \sum_{n=o+1}^{q} l(\epsilon, r_n). && \text{(A.46)} \end{aligned}$$

These equations follow from the definition of α' in (A.1) to (A.4). The argument for (A.44) and (A.46) is an inductive one; for (A.44), the start node

$s(a) = b(\mathcal{L})$ must be handled as a special case. Adding (A.44) to (A.46),

$$\alpha'_R(a,q) \leq \alpha'_R(s(a),p) + \sum_{n=p+1}^{o-1} l(\epsilon, r_n) + l(w(a), o) + \sum_{n=o+1}^{q} l(\epsilon, r_n)$$

and substituting in (A.41),

$$\alpha'_R(a,q) \leq \alpha'_R(\mathrm{s}(a),p) + L((w(a)), R_{p+1}^q). \tag{A.47}$$

But by assumption, in (A.34), $\alpha'_R(a,q) > \tilde{\alpha}_R(a,q,\mathcal{A})$, and substituting Eq. (A.40) for $\tilde{\alpha}_R(a,q,\mathcal{A})$ into this we get the left hand side of:

$$\begin{pmatrix} \tilde{\alpha}_R(\mathrm{s}(a),p,\mathcal{A}) \\ +L((w(a)), R_{p+1}^q) \end{pmatrix} < \alpha'_R(a,q) \leq \begin{pmatrix} \alpha'_R(\mathrm{s}(a),p) \\ +L((w(a)), R_{p+1}^q) \end{pmatrix}$$

(the right hand side is (A.47)), which implies $\tilde{\alpha}_R(\mathrm{s}(a),p,\mathcal{A}) < \alpha'_R(\mathrm{s}(a),p)$, which is an earlier pair for which (A.34) holds, which is a contradiction.

C) x is the start node $x = b(\mathcal{L})$. This is the easiest case. As with Lemma 1, we can directly prove (A.32) with equality using the definitions of $\alpha'_R(b(\mathcal{L}),q)$ in (A.3) and (A.4), and the definition of L in (A.12). Recall that for the start node, the set $S : \{S \in b(\mathcal{L})\}$ contains one member consisting of the empty sequence $\emptyset$, and $p(\emptyset) = 1$. We will not fill in the details of this. □

Proof of Theorem 1. Theorem 1 (Eq. (A.15)) can be expressed using (A.5) and (A.14), as:

$$\alpha'_R(\mathrm{e}(\mathcal{L}), |R|) = \min_{\mathcal{A}} \frac{\sum_{S\in\mathrm{e}(\mathcal{L})} p(S) L(W(S), R, \mathcal{A})}{\sum_{S\in\mathrm{e}(\mathcal{L})} p(S)}. \tag{A.48}$$

Using Lemma 1 for the end node and position $q = |R|$,

$$\alpha'_R(\mathrm{e}(\mathcal{L}), |R|) \geq \frac{\sum_{S\in\mathrm{e}(\mathcal{L})} p(S) L(W(S), R, \hat{\mathcal{A}})}{\sum_{S\in\mathrm{e}(\mathcal{L})} p(S)}, \tag{A.49}$$

and since this is true for a particular alignment $\hat{\mathcal{A}}$, it must also be true if we take the minimum over the right hand side for all $\mathcal{A}$. This establishes (A.48) with $\geq$ in place of $=$. Taking Lemma 2 for the end node and position $q = |R|$ gives us:

$$\alpha'_R(\mathrm{e}(\mathcal{L}), |R|) \leq \min_{\mathcal{A}} \frac{\sum_{S\in\mathrm{e}(\mathcal{L})} p(S) L(W(S), R, \mathcal{A})}{\sum_{S\in\mathrm{e}(\mathcal{L})} p(S)}, \tag{A.50}$$

which is the opposite bound. Therefore (A.48) must be true with equality. □

Appendix B. Proof of convergence

The basic result we will prove in Theorem 2 is similar to the kind of bound obtained for E-M algorithms. It says that the quantity we are minimizing, $\hat{L}(\mathcal{L}, R)$, is guaranteed to decrease on each iteration by at least as much as the auxiliary function $\mathcal{Q}(R, \mathcal{L}, R')$ decreases, where R' is the value of the word-sequence that was used to accumulate the statistics. However this leads to a weaker statement about convergence than in E-M, because unlike E-M there is no sense in which our algorithm is guaranteed to reach a local minimum; rather, we guarantee that on no iteration will the auxiliary function increase. Note that we prove the convergence for arbitrary δ, not just for $\delta = 0$.

We define the auxiliary function $\mathcal{Q}(R, \mathcal{L}, R')$ for $|R| = |R'|$, as:

$$\mathcal{Q}(R, \mathcal{L}, R') = \sum_{q=1}^{|R'|} \sum_{s} \gamma_{R'}(q, s) l(s, r_q), \tag{B.1}$$

where $\gamma_{R'}(q, s)$ are the statistics accumulated by our algorithm from the lattice $\mathcal{L}$ and the word-sequence R'.

Theorem 2.

$$\mathcal{Q}(R, \mathcal{L}, R') - \mathcal{Q}(R', \mathcal{L}, R') \geq \hat{L}(\mathcal{L}, R) - \hat{L}(L, R'). \tag{B.2}$$

Note that in the application of Theorem 2 to convergence of our algorithm, R' will be the word-sequence that is used to compute statistics on a particular iteration, and R is considered as a free variable. We will choose R on each iteration to minimize $\mathcal{Q}(R, \mathcal{L}, R')$.

Proof of Theorem 2. Let K be the total number of pairs (x, q) where x is an arc or a node and $0 \leq q \leq |R|$. We will construct a sequence of functions $\mathcal{Q}^{(k)}(R, \mathcal{L}, R')$ for $0 \leq k \leq K$, where (B.2) holds for each $\mathcal{Q}$ in the sequence, i.e.

$$\mathcal{Q}^{(k)}(R, \mathcal{L}, R') - \mathcal{Q}^{(k)}(R', \mathcal{L}, R') \geq \hat{L}(\mathcal{L}, R) - \hat{L}(L, R') \tag{B.3}$$

and will show that the last element of this sequence, $\mathcal{Q}^{(K)}(\ldots)$, is equal to the auxiliary function of (B.1); this implies (B.2).

Note that this sequence of functions is a mathematical device only used in the proof, and the index k does not correspond to any index in our algorithms.

Each index $k > 0$ is associated with a pair, in reverse order with respect to the partial ordering introduced in (A.19). Let p_k be the pair (x, q) associated with index k, so that $p_1 = (\mathrm{e}(\mathcal{L}), |R|)$ and $p_K = (\mathrm{s}(\mathcal{L}), 0)$. The first index $k = 0$ is not assicated with a pair. In this proof, we will also define quantities ${\beta'}^{(k)}_{R'}(x,q)$ and $\gamma^{(k)}_{R'}(q,s)$; there is a correspondence between these and the β' and γ quantities in the algorithm of Figure 5. The γ quantities on iteration K will be identical to the γ quantities accumulated in that algorithm. The correspondence for the β' quantities is a little more complicated, because to simplify the proof, we set the β' for (x,q) to zero on the iteration corresponding to that pair. We do not do this in the algorithm because those values are never accessed again. (It would be possible to reformulate the algorithm with β' as a queue rather than an array, to take advantage of this). The correspondence is that if $p_k = (x,q)$, then ${\beta'}^{(k-1)}_{R'}(x,q)$ equals the final value of $\beta'(x,q)$ in the algorithm, where we interpret $\beta'_{arc}(q)$ in the algorithm as $\beta'(a,q)$ for the current arc being processed. These statements can be verified by the reader.

Throughout the proof, the distinction between the quantities $\alpha'_{R'}(x,q)$ and $\alpha'_R(x,q)$ is crucial. Think of the former as a specific quantity, evaluated for a known word sequence R', and think of the latter as a function of arbitrary R.

We define the auxiliary function at each step $0 \leq k \leq K$ as:

$$\begin{aligned}\mathcal{Q}^{(k)}(R,\mathcal{L},R') \;:=\; & \sum_{x,q} {\beta'}^{(k)}_{R'}(x,q)\alpha'_R(x,q) \\ & + \sum_{q=1}^{|R|}\sum_{s}\gamma^{(k)}_{R'}(q,s)l(s,r_q). \end{aligned} \tag{B.4}$$

We want the first auxiliary function in the sequence to be the same as $\hat{L}(\mathcal{L},R)$, and this can be accomplished by setting

$$\gamma^{(0)}_{R'}(q,s) \;:=\; 0 \tag{B.5}$$

$${\beta'}^{(0)}_{R'}(x,q) \;:=\; 0,\; x \neq \mathrm{e}(\mathcal{L}) \vee q \neq |R| \tag{B.6}$$

$${\beta'}^{(0)}_{R'}(\mathrm{e}(\mathcal{L}),|R|) \;:=\; 1. \tag{B.7}$$

From the above, (B.4) and (A.5), we can see that

$$\mathcal{Q}^{(0)}(R,\mathcal{L},R') = \alpha'_R(\mathrm{e}(\mathcal{L}),|R|) = \hat{L}(\mathcal{L},R). \tag{B.8}$$

It follows from this that (B.3) holds with equality for $k = 0$. We proceed with an inductive argument. For $1 \leq k \leq K$, we will show that

$$\left(\begin{array}{c} \mathcal{Q}^{(k)}(R,\mathcal{L},R') \\ -\mathcal{Q}^{(k)}(R',\mathcal{L},R') \end{array}\right) \geq \left(\begin{array}{c} \mathcal{Q}^{(k-1)}(R,\mathcal{L},R') \\ -\mathcal{Q}^{(k-1)}(R',\mathcal{L},R') \end{array}\right). \tag{B.9}$$

Adding this to (B.3) for $k-1$, establishes (B.3) for k. Doing this for $1 \leq k \leq K$ proves (B.3) for K. Finally, we will establish that $\mathcal{Q}^{(K)}(R, \mathcal{L}, R')$ is the same as $\mathcal{Q}(R, \mathcal{L}, R')$.

For convenience, we define the β' and γ quantities for $k > 0$ at the same time as we prove (B.9). Since these quantities are mostly the same for successive values of k, it is easiest to define them by stating that $\gamma_{R'}^{(k+1)}(q,s) = \gamma_{R'}^{(k)}(q,s)$ and ${\beta'}_{R'}^{(k+1)}(x,q) = {\beta'}_{R'}^{(k)}(x,q)$ unless we state otherwise below. We will introduce any exceptions to this rule with (†). For each $1 \leq k \leq K$, if $p_k = (x,q)$ then (†)

$$ {\beta'}_{R'}^{(k)}(x,q) := 0, \qquad \text{(B.10)} $$

i.e. we set to zero the β' quantity for a pair (x,q) as we process that pair. We remark that for any k, x and q,

$$ {\beta'}_{R'}^{(k)}(x,q) \geq 0. \qquad \text{(B.11)} $$

This can be verified by noting that the β' quantities are nonnegative for $k = 0$, and by checking all the equations introduced with (†) to verify that no step can introduce a negative quantity.

Now we show that for any value of $p_k = (x,q)$, (B.9) holds. There are various cases.

A) x is a non-initial node $n = x$. In this case, for each $a \in \mathrm{pre}(n)$ we do (†)

$$ {\beta'}_{R'}^{(k)}(a,q) := {\beta'}_{R'}^{(k-1)}(a,q) + {\beta'}_{R'}^{(k-1)}(n,q)\frac{\alpha(a)}{\alpha(n)}. \qquad \text{(B.12)} $$

It follows from the definition of $\mathcal{Q}$ in (B.4) and from (B.10) and (B.12) that

$$ \begin{matrix} \mathcal{Q}^{(k)}(R,\mathcal{L},R') \\ -\mathcal{Q}^{(k-1)}(R,\mathcal{L},R') \end{matrix} = {\beta'}_{R'}^{(k-1)}(n,q) \begin{pmatrix} -{\alpha'}_R(n,q) \ + \\ \sum_{a\in\mathrm{pre}(n)} {\alpha'}_R(a,q)\frac{\alpha(a)}{\alpha(n)} \end{pmatrix} \qquad \text{(B.13)} $$

and given Eq. (A.2) it is easy to see that the parenthesis in the r.h.s. of Eq. (B.13) is zero. This establishes that $\mathcal{Q}^{(k)}(R,\mathcal{L},R') = \mathcal{Q}^{(k-1)}(R,\mathcal{L},R')$, and (B.9) follows with equality.

B) x is the initial node $n = b(\mathcal{L}) = x$.

i) $q > 0$. In this case, (†)

$$ \begin{aligned} {\beta'}_{R'}^{(k)}(n,q-1) &:= {\beta'}_{R'}^{(k-1)}(n,q-1) + {\beta'}_{R'}^{(k-1)}(n,q) && \text{(B.14)} \\ \gamma_{R'}^{(k)}(q,\epsilon) &:= \gamma_{R'}^{(k-1)}(q,\epsilon) + {\beta'}_{R'}^{(k-1)}(n,q). && \text{(B.15)} \end{aligned} $$

It follows from the above, and (B.4) and (B.10), that:

$$\begin{pmatrix} \mathcal{Q}^{(k)}(R,\mathcal{L},R') \\ -\mathcal{Q}^{(k-1)}(R,\mathcal{L},R') \end{pmatrix} = {\beta'}^{(k-1)}_{R'}(n,q) \begin{pmatrix} -\alpha'_R(n,q) \\ +\alpha'_R(n,q-1) \\ +l(\epsilon,r_q) \end{pmatrix}. \tag{B.16}$$

From (A.4) with $n = b(\mathcal{L})$, the parenthesis on the r.h.s. is zero, and again (B.9) follows with equality.

ii) $q = 0$. In this case there are no further changes to β' or γ other than (B.10). This implies that $\mathcal{Q}^{(k)}(R,\mathcal{L},R') = \mathcal{Q}^{(k-1)}(R,\mathcal{L},R')$ since only ${\beta'}^{(k)}_{R'}(b(\mathcal{L}),0)$ changed, and $\alpha'_{R'}(b(\mathcal{L}),0) = 0$ by (A.3). Again, (B.9) follows with equality.

C) x is an arc $a = x$. We consider three cases below. The text in the descriptions of the cases are for intuition only. If more than one apply (in case of a tie in the α' recursion), the proof holds through whichever argument we choose.

i) One symbol consumed, $\alpha'_{R'}(a,q) = \alpha'_{R'}(\mathrm{s}(a),q-1) + l(w(a),r'_q)$. This is the top case of the α' recursion of (A.1). We change β' and γ with (†)

$$\begin{aligned} {\beta'}^{(k)}_{R'}(\mathrm{s}(a),q-1) &:= {\beta'}^{(k-1)}_{R'}(\mathrm{s}(a),q-1) + {\beta'}^{(k-1)}_{R'}(a,q) && \text{(B.17)} \\ \gamma^{(k)}_{R'}(q,w(a)) &:= \gamma^{(k-1)}_{R'}(q,w(a)) + {\beta'}^{(k-1)}_{R'}(a,q). && \text{(B.18)} \end{aligned}$$

It follows from the above, and (B.10) and (B.4), that

$$\begin{pmatrix} \mathcal{Q}^{(k)}(R,\mathcal{L},R') \\ -\mathcal{Q}^{(k-1)}(R,\mathcal{L},R') \end{pmatrix} = {\beta'}^{(k-1)}_{R'}(a,q) \begin{pmatrix} -\alpha'_R(a,q) + l(w(a),r_q) \\ +\alpha'_R(\mathrm{s}(a),q-1) \end{pmatrix}. \tag{B.19}$$

Let us write $f(R)$ and $g(R)$ for the left and right hand sides of (B.19). The statement $f(R) \geq f(R')$ is equivalent to (B.9). Thus, if we show that $g(R) \geq g(R')$ then we prove (B.9). By (B.11) the β' quantities are nonnegative, so it is sufficient to show that

$$\begin{pmatrix} \alpha'_R(\mathrm{s}(a),q-1) \\ +l(w(a),r_q) - \alpha'_R(a,q) \end{pmatrix} \geq \begin{pmatrix} \alpha'_{R'}(\mathrm{s}(a),q-1) \\ +l(w(a),r'_q) - \alpha'_{R'}(a,q) \end{pmatrix}.$$

The condition for this case to apply is that $\alpha'_{R'}(a,q) = \alpha'_{R'}(\mathrm{s}(a),q-1) + l(w(a),r'_q)$, so the r.h.s. of the equation above is zero. It follows from (A.1) that the l.h.s. is nonnegative.

ii) No symbol consumed, $\alpha'_{R'}(a,q) = \alpha'_{R'}(\mathrm{s}(a),q) + l(w(a),\epsilon) + \delta$ This is the middle case of the α' recursion of (A.1). In this case, (†)

$$\beta'^{(k)}_{R'}(\mathrm{s}(a),q) \;\; := \;\; \beta'^{(k-1)}_{R'}(\mathrm{s}(a),q) + \beta'^{(k-1)}_{R'}(a,q). \tag{B.20}$$

From (B.20), (B.10) and (B.4),

$$\begin{pmatrix} \mathcal{Q}^{(k)}(R,\mathcal{L},R') \\ -\mathcal{Q}^{(k-1)}(R,\mathcal{L},R') \end{pmatrix} = \beta'^{(k-1)}_{R'}(a,q) \begin{pmatrix} -\alpha'_R(a,q) \\ +\alpha'_R(\mathrm{s}(a),q) \end{pmatrix} \tag{B.21}$$

Following similar steps to those between (B.19) and (B.20), it is sufficient to prove that:

$$\alpha'_R(\mathrm{s}(a),q) - \alpha'_R(a,q) \geq \alpha'_{R'}(\mathrm{s}(a),q) - \alpha'_{R'}(a,q). \tag{B.22}$$

The condition for being in this case is $\alpha'_{R'}(a,q) = \alpha'_{R'}(\mathrm{s}(a),q) + l(w(a),\epsilon) + \delta$. Adding this to (B.22) and rearranging,

$$\alpha'_R(\mathrm{s}(a),q) + l(w(a),\epsilon) + \delta \geq \alpha'_R(a,q). \tag{B.23}$$

This follows from the α' recursion of (A.1).

iii) Multiple symbols consumed, $\alpha'_{R'}(a,q) = \alpha'_{R'}(a,q{-}1) + l(\epsilon, r'_q)$. This is the bottom line of (A.1). Here (†),

$$\begin{aligned} \beta'^{(k)}_{R'}(a,q{-}1) \;\; &:= \;\; \beta'^{(k-1)}_{R'}(a,q{-}1) + \beta'^{(k-1)}_{R'}(a,q) && \text{(B.24)} \\ \gamma^{(k)}_{R'}(q,\epsilon) \;\; &:= \;\; \gamma^{(k-1)}_{R'}(q,\epsilon) + \beta'^{(k-1)}_{R'}(a,q). && \text{(B.25)} \end{aligned}$$

From the above, (B.10) and (B.4),

$$\begin{pmatrix} \mathcal{Q}^{(k)}(R,\mathcal{L},R') \\ -\mathcal{Q}^{(k-1)}(R,\mathcal{L},R') \end{pmatrix} = \beta'^{(k-1)}_{R'}(a,q) \begin{pmatrix} -\alpha'_R(a,q) \\ +\alpha'_R(a,q{-}1) \\ +l(\epsilon,r_q) \end{pmatrix}. \tag{B.26}$$

Following the same steps as used after (B.19), it suffices to prove that:

$$\begin{pmatrix} \alpha'_R(a,q{-}1) \\ +l(\epsilon,r_q) - \alpha'_R(a,q) \end{pmatrix} \geq \begin{pmatrix} \alpha'_{R'}(a,q{-}1) \\ +l(\epsilon,r'_q) - \alpha'_{R'}(a,q) \end{pmatrix}. \tag{B.27}$$

It follows from the condition for this case to apply ($\alpha'_{R'}(a,q) = \alpha'_{R'}(a,q{-}1) + l(\epsilon,r'_q)$), that the r.h.s. is zero. The recursion for α' of (A.1) shows that the l.h.s. is nonnegative.

We have now handled all the possible cases, which completes the proof of (B.3). It remains to show that

$$\mathcal{Q}(R, \mathcal{L}, R') = \mathcal{Q}^{(K)}(R, \mathcal{L}, R'). \tag{B.28}$$

We have previously stated that $\gamma_{R'}(q, s) = \gamma^{(K)}_{R'}(q, s)$, where $\gamma_{R'}(q, s)$ is the $\gamma(q, s)$ quantity from the algorithm of Figure (5). The reader can verify that ${\beta'}^{(K)}_{R'}(x, q) = 0$ for any (x, q), since each step k zeroes out its own value and only increments values for $k' > k$. Eq. (B.28) follows from the above two observations and from the definitions of $\mathcal{Q}$ and $\mathcal{Q}^{(k)}$ in (B.1) and (B.4). □

Now we can directly address the convergence properties of the decoding algorithm shown in Figure 6.

Lemma 3. *The auxiliary function $\mathcal{Q}(R, \mathcal{L}, R')$ of* (B.1) *can be minimized over R by setting for each q*

$$r_q \leftarrow \max_s \gamma(q, s). \tag{B.29}$$

We will not offer a proof of this as it is trivial.

Corollary 2. *In the decoding algorithm of Figure 6, the approximated edit distance $\hat{L}(\mathcal{L}; R)$ will not increase on any iteration, and the algorithm will terminate.*

Proof of Corollary 2. It follows from Theorem 2 that the decrease in $\hat{L}(\mathcal{L}; R)$ must equal or exceed the decrease in auxiliary function $\mathcal{Q}(R, \mathcal{L}, R')$. Since we exactly minimize the auxiliary function on each iteration, the change in $\mathcal{Q}$ must be zero or negative. If it is zero, we terminate. If it is negative, we must decrease $\hat{L}(\mathcal{L}; R)$ by at least that much. The algorithm will terminate in a finite number of iterations because the number of (non-ϵ) words in the word-sequence on any iteration is bounded by twice the average length of paths through the lattice (we can show this via an argument on the sum of statistics γ for non-ϵ words), and words in R must be picked from the words in the lattice, giving a finite number of possible word-sequences; it follows that there cannot be an infinite sequence of word-sequences, each of which gives a lower $\hat{L}$ than the other. □

Appendix C. Extensions of the algorithm

Appendix C.1. Extension to lattices with multiple start and end nodes

Up to now we have assumed that lattices have a single start and end node, as this simplifies the algorithms and the theory. It is easy to generalize to lattices with multiple start and end nodes, by adding "dummy" arcs a with $p(a) = 1$ and $w(a) = \epsilon$, connecting to new unique start and end nodes. If this is not convenient, for the most part the algorithms will work by simply making the statements that apply to the unique start and end node, apply instead to all start and end nodes. For example, in the edit distance computation (Figure 3), when we write $\beta(N) \leftarrow 1$, we would write instead $\forall n \in e(\mathcal{L}),\ \beta(n) \leftarrow 1$. There are a few additional changes that would have to be made, apart from these obvious changes:

- In Figure 3, replace $\alpha(N)$ in Line 11 with $\sum_{n\in e(\mathcal{L})} \alpha(n)$.
- In Figure 4, replace Line 23 with: $\hat{L} \leftarrow \frac{\sum_{n\in e(\mathcal{L})} \alpha'(n,Q)\alpha(n)}{\sum_{n\in e(\mathcal{L})} \alpha(n)}$.
- In Figure 5, replace Line 11 with: $\forall n \in e(\mathcal{L}),\ \beta'(n, Q) \leftarrow \frac{\alpha(n)}{\sum_{n\in e(\mathcal{L})\alpha(n)}}$.
- In Figure 5, insert a loop around Lines 29 to 34, covering all start nodes.

Appendix C.2. Time information

In some situations it is necessary to annotate the output with time information: for instance, when scoring with NIST's `sclite` tool. This is easy to incorporate into our algorithm. We do this by accumulating statistics on the times of the arcs. It is possible to do this by computing either the start or end times or both, indexed by word and position or just by position. We chose to store both start and end times, indexed by word and position. We use two associative arrays $\tau_b(q, s)$ and $\tau_e(q, s)$ to store statistics on the beginning and ending times of arcs. We suppose that the time of a node n is accessible as $t(n)$; regardless of the lattice format, this should be possible to obtain as long as the lattice contains consistent timing information. After Line 23 of the statistics accumulation (Figure 5), to accumulate the average start time we would add a line:

$$\textbf{switch } (b_{arc}(q)) \begin{Bmatrix} 1: \tau_b(q, w(a)) \mathrel{+}= t(\mathrm{s}(a))\beta'_{arc}(q) \\ 3: \quad \tau_b(q, \epsilon) \mathrel{+}= t(\mathrm{s}(a))\beta'_{arc}(q) \end{Bmatrix}$$

followed by the same command for τ_e, with $\mathrm{e}(a)$ in place of $\mathrm{s}(a)$. For the

start node, after Line 33 we would add the commands:
$\tau_b(q, \epsilon) \mathrel{+}= t(b(\mathcal{L}))\beta'_{arc}(b(\mathcal{L}))$
$\tau_e(q, \epsilon) \mathrel{+}= t(b(\mathcal{L}))\beta'_{arc}(b(\mathcal{L}))$
Then in the update phase, after picking the most likely word $\hat{r}$ at the current position q in Line 7 of Figure 6, we could work out its start and end times using the expressions $\tau_b(q, \hat{r})/\gamma(q, \hat{r})$ and $\tau_e(q, \hat{r})/\gamma(q, \hat{r})$. These are averaged figures so rounding may be necessary. If storing only (say) start times then the end times can be inferred from the start times of the next word, although this may mean incorporating silence times into real words. There are no theoretical guarantees that these times are consecutively ordered so further post-processing may be necessary for some purposes.

[1] Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., Mohri, M., 2007. Openfst: A general and efficient weighted finite-state transducer library. In: Proc. CIAA. pp. 11–23.

[2] Chang, E., Shi, Y., Zhou, J., , Huang, C., 2001. Speech Lab in a Box: A Mandarin Speech Toolbox to Jumpstart Speech Related Research. In: Proc. Interspeech.

[3] D. Povey, L. Burget, M. Agarwal, P. Akyazi, F. Kai, A Ghoshal, O. Glembek, N. K. Goel, M. Karafiát, A. Rastrow, R. C. Rose, P. Schwarz and S. Thomas, 2010. The Subspace Gaussian Mixture Model – a Structured Model for Speech Recognition. Submitted to: Computer Speech and Language.

[4] Evermann, G., Woodland, P., 2000. Posterior probability decoding, confidence estimation and system combination. In: Proc. Speech Transcription Workshop.

[5] Fiscus, J. G., 1997. A post-processing system to yield reduced word error rates: Recognizer Output Voting Error Reduction (ROVER). In: ASRU.

[6] Goel, V., Byrne, W., 2000. Minimum Bayes-risk automatic speech recognition. Computer Speech and Language 14 (2).

[7] Graeme Blackwood, Adrià de Gispert and William Byrne, 2010. Efficient Path Counting Transducers for Minimum Bayes-Risk Decoding of Statistical Machine Translation Lattices. In: Proc. ACL.

[8] Heigold G., Macherey W., Schlüter R., Ney H., 2005. Minimum Exact Word Error Training. In: Proc. ASRU.

[9] Hermansky, H., 1990. Perceptual linear predictive (PLP) analysis of speech. Journal of the Acoustical Society of America 87, 1738–1752.

[10] Hoffmeister, B., Schlüter, R., Ney, H., 2009. Bayes Risk Approximations Using Time Overlap with an Application to System Combination. In: Proc. Interspeech.

[11] Kirpatrick, S., Gelatt, Jr., C., Vecchi, M., May 1983. Optimization by simulated annealing. Science 220, 671–680.

[12] L. Mangu, E. Brill and A. Stolcke, 2000. Finding Consensus in Speech Recognition: Word Error Minimization and Other Applications of Confusion Networks. Computer Speech and Language 14 (4).

[13] Levenshtein, V. I., 1966. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10.

[14] Mohri, M., Pereira, F., Riley, M., 2002. Weighted finite-state transducers in speech recognition. Computer Speech and Language 20 (1), 69–88.

[15] Papineni, K., Roukos, S., Ward, T., Zhu, W.-J., 2002. Bleu: a method for automatic evaluation of machine translation. In: Proc. ACL. pp. 311–318.

[16] Povey, D., Burget, L., Agarwal, M., Akyazi, P., Feng, K., Ghoshal, A., Glembek, O., Goel, N. K., Karafiát, M., Rastrow, A., Rose, R. C., Schwarz, P., Thomas, S., 2010. Subspace Gaussian Mixture Models for Speech Recognition. In: Proc. ICASSP.

[17] Povey, D., Kanevsky, D., Kingsbury, B., Ramabhadran, B., Saon, G., Visweswariah, K., 2008. Boosted MMI for Feature and Model Space Discriminative Training. In: Proc. ICASSP.

[18] Povey, D., Woodland, P., 2002. Minimum Phone Error and I-smoothing for Improved Discriminative Training. In: Proc. ICASSP.

[19] Qian, Y., Lin, S., Zhang, Y., Liu, Y., Liu, H., Liu, Q., 2004. An Introduction to Corpora Resources of 863 Program for Chinese Language

Processing and Human-Machine Interaction. In: Proc. ALR-04, affiliated to IJCNLP.

[20] Saon, G., Soltau, H., Chaudhari, U., Chu, S., Kingsbury, B., Kuo, H.-K., Mangu, L., Povey, D., 2010. The IBM 2008 GALE Arabic Speech Transcription System. In: Proc. ICASSP.

[21] Stolcke, A., König, Y., Weintraub, M., 1997. Explicit word error minimization in N-best list rescoring. In: 5th European Conf. on Speech Comm. and Technology. pp. 163–166.

[22] Tian, Y., Zhou, J., Lin, H., Jiang., H., 2006. Tree-based Covariance Modeling of Hidden Markov Models. IEEE Trans. on Audio, Speech and Language processing 14 (6), 2134–2146.

[23] Wessel, F., Schlüter, R., H., N., 2001. Explicit word error minimization using word hypothesis posterior probabilities. In: Proc. ICASSP.

[24] Xu, H., Povey, D., Mangu, L., Xhu, J., 2010. An Improved Consensus-Like method for Minimum Bayes Risk Decoding and Lattice Combination. In: Proc. ICASSP.

[25] Xu H., Povey D., Zhu J. and Wu G., 2009. Minimum Hypothesis Phone Error as a Decoding Method for Speech Recognition. In: Proc. Interspeech.

[26] Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., Woodland, P., 2009. The HTK Book (for version 3.4). Cambridge University Engineering Department.

[27] Zhang, J., Zheng, F., Li, J., Luo, C., Zhang, G., 2001. Improved Context-Dependent Acoustic Modeling for Continuous Chinese Speech Recognition. In: Proc. Interspeech.