SIMULTANEOUS SPEECH SEGMENTATION AND PHONEME RECOGNITION USING DYNAMIC PROGRAMMING

Raminder S. Bajwa1  Robert M. Owens2  Thomas P. Kelliher3

1Semiconductor Research Laboratory, Hitachi America Ltd., 201 East Tasman Drive, San Jose CA 95134.
2Department of Computer Science & Engineering, Pennsylvania State University, University Park, PA., 16802.
3Department of Mathematics and Computer Science at Westminster College, New Wilmington PA 16172.

ABSTRACT

In this paper a dynamic programming algorithm for simultaneous speech segmentation and phoneme recognition is presented. Given a sequence of samples of an unknown speech pattern and a library of phonemes, this algorithm finds the best phonological match and, with a backtracking step, identifies the phoneme boundaries. This approach is different from a traditional two step process whereby first the phoneme boundaries are determined locally and then speech recognition is performed. Its advantage over the two step process is that incorrect phoneme boundaries due to slurring or sudden changes in the speech are reduced. Unlike other dynamic programming algorithms, it does not lend itself to systolic wavefront processing, hence an alternate parallel algorithm is presented.

1. INTRODUCTION

Dynamic programming in the form of dynamic time warping has been applied extensively in speech recognition [WBA83] and other related areas of pattern matching. Typically the problem is formulated such that the unknown speech pattern must be checked against a large library of words. The computation required depends directly on the size of the library. In order to reduce the complexity of the task, the words in the vocabulary are broken down into atomic units which are often phonemes.

Dynamic time warping is applied to strings of phonemes to effect a recognition. This entails that the speech pattern being tested be segmented into phonemes a priori [PF90, SC78]. The algorithm presented in this paper treats the two problems of segmentation and recognition in a unified manner. The combined problem was first formulated in a paper by Bridle and Sedgwick [BS77] but only a restricted version of the problem was solved in their paper. The restricted problem was: given a sequence of samples representing a segmented word template and a sequence of samples representing an unknown word find the cost of the best mapping of the template onto the given unknown word. The best mapping also implicitly resulted in a segmentation of the unknown word corresponding to the segments of the template.

The algorithm presented here solves the following problem: given a sequence of samples representing continuous speech and a library of phonemes, find the best phonological match and the corresponding phonetic segmentation. The phoneme boundaries are identified after the recognition process is complete by a backtracking step.

Note that Bridle and Sedgwick’s algorithm [BS77] cannot be trivially extended to include the one presented here. Unlike their word templates which are segmented a priori the templates used in this paper are not segmented (they are phonemes and hence atomic). Also, if the notion of a “word” is extended to include speech sequences corresponding to sentences then the number of templates needed is proportional to the number of possible sentences that can be recognized resulting in a very large library. For the algorithm presented here the size of the library is fixed and is comprised of all the phonemes.

Speech segmentation based on phoneme recognition can be used as a preliminary step in word recognition based on phonemes, as it reduces the size of the search space for Markoff models, hence reducing their learning time. Experiments were conducted on utterances taken from the TIMIT Corpus [GIL93]. For the discussion in this paper, it is assumed that parameterized speech rather than raw speech is available; in general raw speech can be converted into an encoded form by using LPC or systems such as CELP coders [MB89].

2. THE SERIAL ALGORITHM

Let the sequence of speech samples to be analyzed be denoted by $S[n]$ and let $P$ denote the set of phonemes. Let $p_k$ denote a member of $P$ and $p_k[n]$ be the sequence of samples which make up $p_k$. It is assumed that both the speech to be analyzed and the phonemes are sampled at a constant and common rate.

![Fig. 1. Progression of the dynamic programming algorithm](image)

Figure 1 shows the flow of the dynamic programming algorithm. Along the $y$-axis are all the phonemes (only four are shown), and along the $x$-axis is the unknown speech pattern. Let $S_i$ denote the value of the global cost, where $i$ and $j$ are indices representing samples along the two axes. For example, in Figure 1, $S_{m,k}$ is the cost of a global match of the interval along the sequence $S[n]$ from samples 0 to
m and at the kth sample along the phoneme axis. Index i increases monotonically, in Si,j while index j does not. Initially S0,p = 0. Computation for S_{ij} begins with j = 0. Starting with an initial cost of zero progress by developing each of the curves b and c in Figure 1 according to the warping function chosen (for simplicity similar curves for the other phonemes have been omitted). When the end of a curve is reached the cost of the match till that point is recorded. Consider the point (m, n) in Figure 1. A new match is started at this point, and an initial cost has to be established. Setting the initial cost to S_{0,n} is sub-optimal in general as this will not provide the best solution. An initial cost of \( C_{m,n} \) is chosen such that,

\[
C_{m,n} = \min_k (S_{m,k})
\]

where \( S_{m,k} \) is the global cost at all positions along column m at which a match with some phoneme \( p_i \) is completed. If \( S_{m,k} \), as shown in Figure 1 happens to be the minimum of the \( S_{m,p} \) values at m then set \( C_{m,n} = S_{m,k} \). Continue in this fashion till the end of the speech sequence is reached. At this point the recognition problem has been solved and the resulting segmentation can be obtained by backtracking from the best final cost.

![Diagram of dynamic time warping lattice](image)

Fig. 2. The dynamic time warp lattice, which is at the core of the dynamic programming algorithm shown in Figure 1.

The template matching process is illustrated in Figure 2. An arbitrary interval \( m \leq i \leq n \), of \( S[n] \) is chosen as the unknown to be matched against \( p_k \). The horizontal axis represents the unknown sequence \( S[n] \) and along the vertical axis is phoneme sequence \( p_k[n] \).

If the cost of a local match from \( (m,n) \) to \( (i,j) \) is denoted as \( \hat{S}_{ij} \) then with reference to Figure 2 it is related to the global cost \( S_{m+i,n+j} \) as

\[
\hat{S}_{ij} = S_{m+i,n+j} - C_{m,n}
\]

which can be written as

\[
S_{m+i,n+j} = (\hat{S}_{p,q} + W_{p,q} D_{m+i,n+j}) + C_{m,n}
\]

in which

\[
\hat{S}_{p,q} + W_{p,q} D_{m+i,n+j}
\]

is the general form of the recursion for dynamic time warping and \( C_{m,n} \) is the best global cost till \( m,n \).

Many different minimization constraints can be used, the simplest could be

\[
S_{m+i,n+j} = D_{m+i,n+j} + \min \left\{ \begin{array}{c}
S_{m+i-1,n+j-1} \\
S_{m+i-1,n+j} \\
S_{m+i,n+j-1}
\end{array} \right\}
\]

or a more complex function such as

\[
S_{m+i,n+j} = \min \left\{ \begin{array}{c}
D_{m+i+1,n+j} + 2 \cdot D_{m+i-1,n+j} + S_{m+i,n+j-1} \\
2 \cdot D_{m+i,n+j} + S_{m+i-1,n+j} \\
D_{m+i,n+j} + 2 \cdot D_{m+i+1,n+j-1} + S_{m+i-1,n+j-2}
\end{array} \right\}
\]

From a numerical standpoint, it is possible that, for long speech sequences the cost function can grow significantly, resulting in overflow. Hence, a simple weighting function is adopted which bounds the growth of the cost function. The weighting is as follows, the cost up to the current point, or the history, is scaled by 0.95 while the cost at the current position is scaled by 0.05. This does not change the behavior of the algorithm, but ensures an upper bound on the cost function. Similarly, to consider underflow assume a threshold such that any time the function dips below this threshold it is assumed that the sequences differ so much that no further computation is needed and hence the cost function does not shrink anymore.

### 3. A PARALLEL ALGORITHM

The algorithm as outlined in the previous section has an obvious serializability, \( C_{m,n} \) must be known before the next phoneme match can proceed. As a result of this, it cannot be parallelized as is. A different algorithm is proposed for parallel implementation.

Define \( C_{p} \) as the cost of aligning phoneme \( p_k \) along an interval \( m \leq k \leq n \) of \( S[n] \). The parallel algorithm has two distinct phases. During the first phase, all possible \( C_{p} \) are determined in parallel. Duplicates are avoided by retaining only the best solutions. Assume that there are more than one paths from \( i \) to \( j \), one along \( C_{p_i} \) and another along \( C_{p_j} \). If \( C_{p_i} < C_{p_j} \) then retain \( C_{p_i} \). All \( C_{p} \) are computed in parallel during this phase. This is accomplished by executing the standard recursion for DTW on a 2-D mesh architecture in a pipelined fashion [WBA83].

The second phase proceeds by defining a directed acyclic graph using the the \( C_{p} \) generated in the first phase and then finding the shortest (least-cost) path in the resultant graph starting at node 0 and ending at node \( N \) where \( N \) is the highest numbered sample in \( S[n] \).

![Diagram of directed acyclic graph](image)

Fig. 3. The \( C_{p} \) computed in Phase 1

### 4. A PARALLEL IMPLEMENTATION

For the parallel implementation, assume that a 2-D mesh connected processor array is available. Further, assume that the unknown speech pattern is fed along one edge of the array while the phonemes are available through an adjacent edge. The process stops when all \( C_{p} \) have been computed.

Phase 2 requires the array to be set up as an adjacency matrix where \( processor_i \) stores the cost of the best path from sample \( i \) to sample \( j \), the identity of the phoneme \( p_k \) which ends at \( j \) corresponding to the best cost, and a path cost \( bestcost \) which is the best cost seen from 0 to \( i \). Each \( processor_i \) also contains a back pointer \( Bak.Ptr \) pointing to the processor from which the phoneme resulting in the best cost at \( processor_j \) originated. Note that not all processors will contain a valid entry. An ordered triplet \( (Start, End, Cost) \) is fanned out and propagated through the
array starting at position \((0, 0)\) as \((0, 0, \_INVAL)\) in a systolic wavefront fashion.

At each processor \(i, j(C_{pi}^*, \text{Bak.Ptr}, \text{Bestcost})\)

\[
\text{if } (\text{End} == i) \text{ and } (\text{Bestcost} > \text{Cost}) \text{ then}
\]

\[
\text{End} = j
\]

\[
\text{Bestcost} = \text{Cost}
\]

\[
\text{Cost} = \text{Cost} + C_{pi}^*
\]

\[
\text{Bak.Ptr} = \text{Start}
\]

\[
\text{Start} = i
\]

\[
\text{endif}
\]

\[
\text{Shift (Start, End, Cost) north}
\]

OR

\[
\text{Shift (Start, End, Cost) east}
\]

On completion, \(\text{processor}_N(C_{iN}^*, \text{Bak.Ptr}, \text{Bestcost})\)

has the best match cost. The segmentation resulting from this match is obtained by backtracking. The last phoneme

started at \((\text{Bak.Ptr}, M)\) and the one before that is obtained from the value of \(\text{Bak.Ptr}\) at \((\text{Bak.Ptr}, M)\) and the current \(\text{Bak.Ptr}\).

5. PERFORMANCE

For the serial algorithm the computation performed at every input sample is \(O(\max_k |p_k| \cdot |P|)\), where \(|P|\) is the number of phonemes and \(\max_k |p_k|\) is the length of the longest phoneme. Hence, the total computational complexity of the serial dynamic programming algorithm is \(O(N \cdot \max_k |p_k| \cdot |P|)\), where \(N\) is the number of samples in the input speech segment. The TIMIT Corpus \([\text{GLF}^+93]\) contains only 45 distinct phonemes; in general the value of \(|P|\) will be bounded by some reasonably small constant. Thus, such an approach would be very effective for real time applications.

In the case of the parallel algorithm the first phase, which comprises of computing all \(C_{pi}^*\)s, can be completed in time \(O(\max_k |p_k| |n|)\) where \(\max_k |p_k| |n|\) is the size of the longest phoneme. The second phase, requires the finding of the lowest cost path in the graph constructed with the \(C_{pi}^*\)s computed in the first phase, completes in \(O(|P|)\) steps. Therefore total cost of the parallel algorithm is \(O(\max_k |p_k| |n| + |P|)\).

This algorithm can be speeded up by using the observation that the adjacency matrix, as defined above, can be quite sparse for large \(N\) in which case one need only store the defined values. If that is done then the time complexity reduces to \(O(\sum_{k=1}^{N} \min_k |p_k|)\) where \(\min_k |p_k|\) is the size of the smallest phoneme. Hence, the time complexity of this version is \(O(\max_k |p_k| |n| + \min_k |p_k| |P|)\).

Table 1 and Table 2 list the results of ten experiments run. The sentences chosen were those considered to be difficult to handle. Under the columns titled “DB” are the phonetic segmentations of utterances as contained in the TIMIT database. Under the columns titled “DP” are the phoneme sequences as derived by the serial dynamic programming algorithm for the corresponding utterances. Figure 3 has a summary of the editing operations and their frequencies for 10 of the experiments conducted.

6. CONCLUSIONS

In this paper a serial dynamic programming algorithm and a parallel algorithm for simultaneous speech recognition and phoneme segmentation are presented. The approach described here differs from previous approaches by treating recognition and segmentation as a single problem and computing them simultaneously. Algorithmically recognition precedes segmentation because the segmentation is identified by a backtracking step performed after the recognition process is complete. Additionally, the segmentation is the one which results from the best recognition solution and hence is a superior segmentation compared to that which results from local phoneme matching. The parallel algorithm is suitable for implementation on VLSI arrays and a mapping is described. Computationally, the dynamic programming method presented in this paper is efficient with time complexity of \(O(\max_k |p_k| \cdot |P|)\) per input for the serial algorithm, where \(|P|\) is the number of phonemes, \(\max_k |p_k|\) is the max number of parameterized samples per phoneme. The two step dynamic time warping algorithm for speech recognition \([\text{PFF}90]\) based on phonemes has a complexity per input sample of \(O(w |P|)\) where \(w\) is the warping distance, if the warp is extended to be as large as the longest of the phonemes then the cost becomes \(O(\max_k |p_k| \cdot |P|)\). So, the algorithm presented here is computationally no worse than the conventional approach for recognition. Its advantage is that speech is segmented as a by-product of the recognition process.

REFERENCES


<table>
<thead>
<tr>
<th>Experiments</th>
<th>Matches</th>
<th>Inserts</th>
<th>Deletes</th>
<th>Insertion Cost</th>
<th>Deletion Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp1</td>
<td>33</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Exp2</td>
<td>32</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp3</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Exp4</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp5</td>
<td>33</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Exp6</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Exp7</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp8</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Exp9</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp10</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1
Recognition and Segmentation obtained by the dynamic programming approach (DP) compared with those obtained from the TIMIT database (DB), Experiments 1 through 5.

<table>
<thead>
<tr>
<th>Experiments</th>
<th>Matches</th>
<th>Inserts</th>
<th>Deletes</th>
<th>Insertion Cost</th>
<th>Deletion Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp1</td>
<td>33</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Exp2</td>
<td>32</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp3</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Exp4</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp5</td>
<td>33</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Exp6</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Exp7</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp8</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Exp9</td>
<td>33</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Exp10</td>
<td>35</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2
Recognition and Segmentation obtained by the dynamic programming approach (DP) compared with those obtained from the TIMIT database (DB), Experiments 6 through 10.