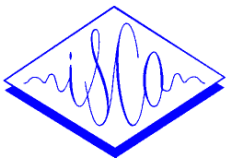

2nd International Conference on
Spoken Language Processing (ICSLP 92)
Banff, Alberta, Canada
October 12-16, 1992

HMM TRAINING ON UNCONSTRAINED SPEECH FOR LARGE VOCABULARY, CONTINUOUS SPEECH RECOGNITION

G. Boulianne, P. Kenny, M. Lennig, D. O'Shaughnessy, P. Mermelstein

INRS-Télécommunications
16, place du Commerce
Verdun (Ile-des-Soeurs), Québec
Canada H3E 1H6

ABSTRACT

Training hidden Markov models for large vocabulary, continuous speech recognition requires large amounts of data. Books on tape are an easily available, very large source of orthographically transcribed speech data. Use of this source of data is problematic for current training algorithms, however, because they require the speech to be first segmented into isolated sentences. In this paper we present a training algorithm to find the maximum likelihood sequence of states in a phonetic HMM model of an unsegmented, unlimited length speech utterance.
The algorithm that we propose has computation time proportional to utterance length, but requires only a fixed amount of memory, independent of utterance length, so that speech input does not have to be segmented into sentences. It considers successive windows on the speech observations. A full Viterbi search is carried on to the end of each window ; then only N paths are retained as the starting paths for the next window. The N survivor paths are not chosen according to their current likelihood, but by looking ahead at their short-time future likelihoods. In practice, we show that N can be reduced to one, while keeping the search optimal, with a limited amount of look-ahead.

10.21437/ICSLP.1992-61

1. INTRODUCTION

A large part of the computation in maximum likelihood estimation of HMM model parameters is spent in search of an optimal alignement between model transitions and speech data. The Viterbi algorithm has interesting properties for this purpose and is most commonly used for the search [1].

Clever implementations of the forward-backward or Viterbi algorithms use previous segmentation knowledge so that their computation time is proportional to the input speech length. Yet their memory needs are still proportional to the speech length, so speech has to be segmented into sentences short enough to fit into a finite memory [2]. Clearly such human intervention is not desirable when dealing with a large amount of training data.

2. A SLIDING WINDOW SEARCH

In the Viterbi algorithm, the final decision about path optimality cannot be made until the end of the utterance. Information about the past has to be stored and this memory requirement grows linearly with time. A *sliding window* algorithm makes a decision before the data has been seen in its entirety:

> Starting with existing ancestor paths at time t, prolong the paths up to time $t + L$ by a Viterbi search. At time $t + L$, choose a limited number N of candidate paths; take these N candidate paths as the ancestors for a new window starting at time $t + L$ and discard the others.

Here L is the window length. Pruning to a constant number of candidates at each window makes memory use constant (computation is still proportional to time). It can also make the search sub-optimal, i.e. not guaranteed to find the Viterbi path for the whole utterance, unless that path can be guaranteed to be among the candidates selected at each window. This general procedure gives rise to a number of variants according to the way the candidate paths are chosen.

One can choose the best path according to a heuristic value computed from the scores. This is in fact the algorithm of [3]. This procedure is never optimal, since there is no way to ensure that the globally optimal path will be chosen at every window.

Another possibility is to prune, at end of window, those paths that have a score which differs from that of the best one by a preset threshold, retaining only the N best paths. This can be viewed as a *global beam-search*: it is like a *beam search* [2] at window boundaries, but still a full search inside the window. Note that although in principle this procedure is not optimal, in practice a beam search can be made as close as desired to optimality by using a large enough threshold [4].

Our previous experience with *two-phone look-ahead* [5], although in the different context of isolated-word recognition, suggested that looking a few phones ahead in the utterance could give valuable information about the optimality of the current partial paths. The next section presents a new pruning rule using this kind of information.

2.1 Looking ahead

In the field of digital communications, Viterbi decoding decisions have often to be made before the end of a message; to get around the problem, the technique of selecting a survivor path by backtracking along the current best path is commonly used [6]. It is based on the observation that partial paths often have a common ancestor, i.e. they merge together at some point in the past.

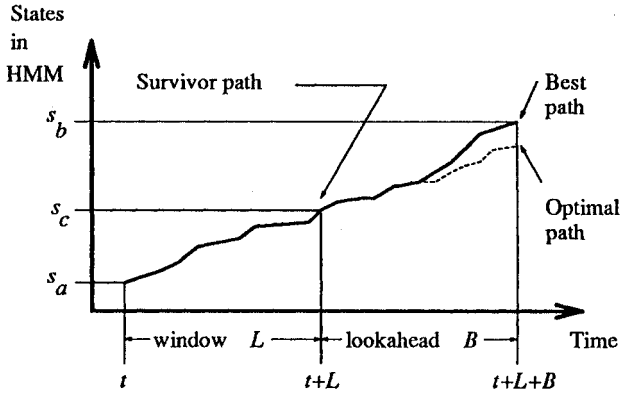

Fig. 2.1 Selection of survivor path by look-ahead.

This suggest the following pruning rule to select candidates at the end of a window, considering not only the current scores, but also short-time future scores. Referring to figure 2.1:

1. Initial conditions: assume an existing window on the observations, covering the interval $[t, t + L]$, and a starting path at node (t, s_a).
2. Search: extend the starting path (using the Viterbi algorithm) into all partial paths ending at time $t + L + B$. Record the partial path having the best score at this time. The corresponding node is $(t + L + B, s_b)$.
3. Survivor selection: backtrack along the path recorded in step 2 to find its ancestor at time $t + L$. This path is selected as the only survivor path. The corresponding node is $(t + L, s_c)$. All other paths can be now discarded.

4. Loop: set a new window beginning at time $t + L$ with the survivor as the starting path and repeat the steps (in other words, set $t = t + L$, $s_a = s_c$ and go to step 1).

For this look-ahead rule to be optimal we only require that the best path and the overall optimal path merge together at some time past $t + L$. In the limit, when B is large enough so that the window spans the entire data file, the search reduces to a full Viterbi search. The search can be made as close to optimal as desired by using a large B. Experimental results will show that convergence of the paths can be obtained for useful values of L and B.

Two properties of the look-ahead search will be used in following sections:

1. While a beam search can also be made close to optimal by making N (number of retained paths) large, a look-ahead search, however, is asymptotically optimal even if only one survivor is kept.

2. The survivor path will be extended into paths that do not necessarily pass through the trellis point used in selecting it. By contrast, in a beam search, only paths following exactly the N retained paths will be explored.

These two properties become especially useful when a look-ahead pruning rule is used in conjunction with the block Viterbi algorithm that will be described in the next section.

2.2 Block Viterbi searching

The *block Viterbi* algorithm [5] computes a Viterbi segmentation directly from the phonetic transcription without explicit reference to the underlying HMM models. In contrast to a simple Viterbi search, as will be seen, duration constraints and approximate segmentation knowledge are readily taken care of by this algorithm. In addition, the two previously stated properties of the look-ahead pruning rule can be used to advantage in the block Viterbi algorithm framework.

2.2.1 The phonetic trellis

Block Viterbi decoding is applied to a *phonetic graph* [5] instead of an HMM model. *Nodes* in the phonetic graph are joined by *branches* and each branch carries a phone label (as opposed to states and transitions in an HMM). Paths in the phonetic graph correspond to sequences of phones; the graph represents all the possible phonetic transcriptions of an utterance. The result of a block Viterbi alignement is thus a segmentation that specifies phones and their entry and exit times.

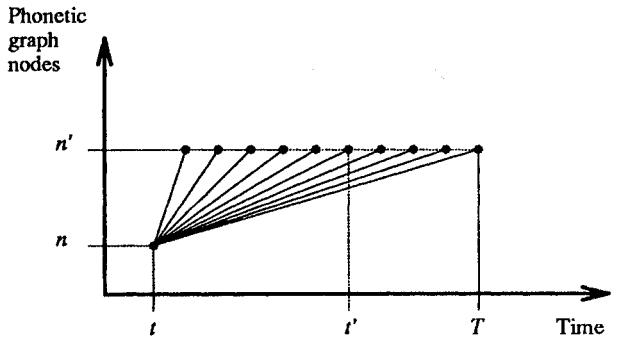

Fig. 2.2 Phonetic treillis for block Viterbi algorithm.

A trellis can be constructed from this graph and a sequence of observations. In Figure 2.2, the point (t, n) corresponds to a time t of the observation sequence and a node n of the phonetic graph. Suppose a phone f joins node n and n' in the phonetic graph. In the trellis, branches labeled f will go from entry time t to all possible exit times t' between times $t+1$ and T. The probabilities of these branches accounting for observations in the interval $[t+1, T]$ are noted $h_t^f(t')$ and called *point scores*.

The total joint probability of a phone sequence and the observations is obtained by multiplying the probabilities of all branches traversed along its path. The optimal path can be found by a breadth-first search of the trellis, i.e. in a Viterbi-like manner. The entry and exit times of each branch lying on the optimal path give the optimal segmentation.

2.2.2 Point scores

We compute point scores of f by replacing the branches labeled f with an HMM model of f. The point score function [5] h_t^f on the interval $[t+1, T]$ is then

$$h_t^f(t') = V(Y_t ... Y_{t'} | f)$$

for entry time t and exit times $t' = t+1, ..., T$, where $V(Y_t ... Y_{t'} | f)$ is the Viterbi score of the observation sequence $Y_t ... Y_{t'}$ given the phone model f. Once the observation sequence is given, this function returns a score for each triplet (phone, entry time, exit time). Duration constraints can be imposed simply by setting $h_t^f(t')$ to zero for certain values of t'; in the case of minimum and maximum durations, for all t' for which $t' - t + 1$ is less than the minimum duration or more than the maximum duration.

Note that point scores can be evaluated efficiently for all t' in an interval $[t + 1, T]$ with only a single trellis calculation.

2.3 Block Viterbi and look-ahead pruning

The sliding window and look ahead principles readily extend to the block Viterbi algorithm in the following way (step numbers here mirror those of section 2.1):

1. Initial conditions: assume a existing window on the observations covering the interval $[t, t + L]$, and a starting partial path ending with a phone f exiting at time at t. (The initial window starts at time 0 at the initial node of the phonetic graph).

1a. Precomputation: compute point scores for all phones in the inventory and for all times in the interval $[t, t + L + B]$.

2. Search: extend the starting path by the block Viterbi algorithm into all possible partial paths ending at time $t + L + B$. Find the partial path having the best score at time $t + L + B$.

3. Backtracking: backtrack along the path found in step 3 until the first phone having its entry time before $t + L$. This phone exits at trellis point (t', n') with $t' \geq t + L$. Record t', n' and the score at point (t', n') ; all other paths can be discarded. (At the end of utterance, the window will be smaller than usual; in this case, choose the best path only among the complete paths and exit the procedure).

4. Loop: repeat from step 2 with a new window extending from t' to $t' + L + B$ and a starting path at t', n'. (At the end of utterance, the window will end sooner than $t' + L + B$).

A beam search constrains the future paths to lie exactly along the retained partial paths. So the N best partial paths have to be chosen not only at time $t + L$, but also inside all the interval around $t + L$ where a phone might exit; otherwise the paths would be constrained to have a phone exiting precisely at $t + L$. The N paths have thus to be chosen among a much larger set of candidate paths than in normal Viterbi searching; in our case the set would be larger by a factor of 50 to 70 (the allowed maximum duration of a phone). Thus a beam search, although natural for normal Viterbi, becomes inefficient for the block Viterbi algorithm.

In contrast, in step 3 of the look-ahead search, only one path needs to be kept even in the block Viterbi algorithm. Note that paths inside the models are not available, so that the best partial path at time $t + L + B$ has to end with a phone exiting precisely at $t + L + B$. This is not catastrophic, however, since it is the ancestor of that path, and not that path itself, that will be retained for the next window (property 2 of section 2.1). The problem becomes less important as B is made larger; it may lead to an error in theory, but not in practice, as long as B is sufficiently big.

3. SEMI-RELAXED TRAINING

We applied the sliding window and block Viterbi algorithm to the problem of finding a segmentation for long sequences of continuous speech training data.

Speech from a book on tape was digitized in files with an average length of 6 minutes (2 or 3 files per chapter). The text transcription of each file was looked up in a pronunciation dictionary to create phonetic graphs. Typical phonetic trellises to be searched for each file had 14 000 nodes and were 36 000 frames long. Experiments were run to show convergence of paths and determine suitable values for window length L, and look-ahead interval B.

In these experiments we were able to reduce computational and memory requirements by using a priori segmentation knowledge (semi-relaxed training [7]). The following section describes how this information can be efficiently incorporated into a sliding window, block Viterbi algorithm with negligible impact on its optimality.

3.1 Use of previous segmentation knowledge

Suppose that we are given an approximate segmentation of the speech data, so that entry and exit times of each segment are within known intervals. Such knowledge could come, for example, from a previous iteration of the training. At any point during the search, the sliding window (plus look ahead interval) will usually include only a subset of the whole phone inventory. Restricting computations to this limited set of models in each window will be referred to as *model pruning*. This kind of pruning can applied to both block Viterbi algorithm and normal Viterbi search.

Entry time pruning is effected by making point scores zero for all entry times outside an interval around the approximate entry time given by the segmentation. In the same way, minimum and maximum durations of phones can be used for *exit time pruning*, by making point scores zero for all exit times outside an interval after entry time. These prunings would be difficult to implement in normal Viterbi search but are straightforward to apply to the Viterbi block algorithm: all that needs to be done is setting appropriate parts of the point scores to zero prior to the search.

Finally, approximate knowledge of the segmentation is used to limit the set of nodes considered in the phonetic graph. This *node pruning* requires, however, that approximate times be assigned to all nodes in the phonetic graph; next section will describe how such an assignement can be computed when an approximate segmentation is available.

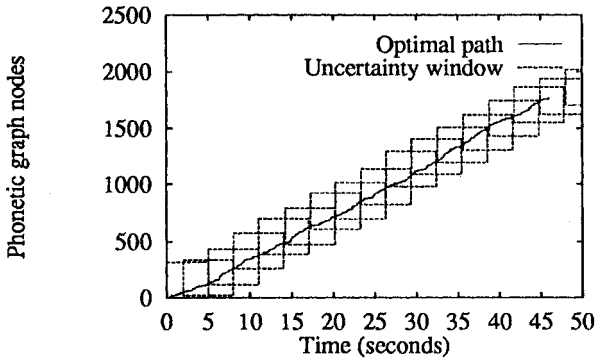

Fig. 3.1 Typical uncertainty window used for node and time pruning.

Since available segmentation data is only approximate, potential entry and exit times are constrained to lie within intervals centered on the estimated entry and exit times. There is a corresponding uncertainty interval for the nodes of the phonetic graph (next section). In this way, for each position of the sliding window during search, the uncertainty and approximate segmentation together define an *uncertainty window*. A large uncertainty window is used so it almost certainly contains the optimal path. Figure 3.1 shows the uncertainty window at different times during search of part of a speech file (which was never seen by the models during training). The optimal path lies comfortably within the uncertainty window.

3.1.1 Node-time assignement

In our experiments multiple pronunciations for each word are allowed, as well as optional silences (and sometimes breath sounds) between words. Thus the phonetic graph has parallel paths. This complicates the use of a priori segmentation information, since the segmentation from a previous iteration, the segmentation found in the current search, and the actual segmentation of the speech may all correspond to different pronunciations. The problem is to find approximate boundary locations for segments that do not appear in the same place, or at all, in the previous segmentation.

The phonetic graph represents all and only the possible pronunciations. Each segmentation then defines a unique path in the phonetic graph, and assigns an entry time to each node on that path. However, some of the nodes, corresponding to some other pronunciations, will have no assigned entry time.

But clearly the segmentation still constrains those nodes: they cannot have an entry occurring before any preceding node in the graph (plus minimum duration of a phone), or later than any succeeding node (minus minimum duration of a phone). Using these constraints, one can assign a minimum and a maximum entry time *to every node* in the phonetic graph.

Figure 3.2 represents a phonetic graph corresponding to the utterance "the forces of". Optional silence loops were inserted, allowing from zero to

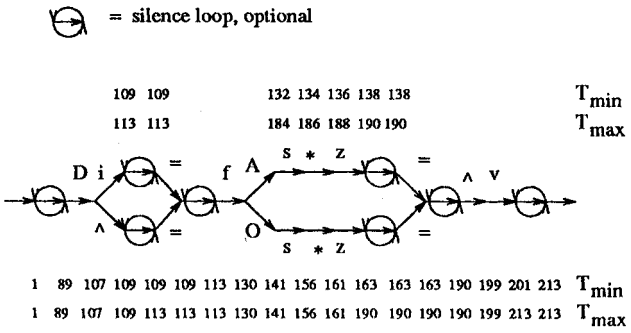

Fig. 3.2 Minimum and maximum entry times for nodes in phonetic graph.

any number of silence phones between words. This way of handling silence is dictated by the need for maximum duration constraints, while still allowing infinite duration for silences.

The figure depicts how entry times are assigned to nodes using an existing segmentation. Nodes lying on the "bottom" path correspond to the existing segmentation. They are assigned equal minimum and maximum entry times (except in loops), as shown by lines of T_{min} and T_{max} under the graph. Other nodes are assigned times based on their neighbours (lines of T_{min} and T_{max} over the graph). The same kind of assignement is used critical path scheduling and PERT for completion times of tasks in a project [8].

Once every node in the phonetic graph has been assigned a minimum and maximum entry time, it is a simple matter to predict which nodes can possibly be visited during a given time interval, and construct the corresponding uncertainty window in the phonetic trellis.

3.2 Experimental setup

3.2.1 Speech processing

Data for experiments consisted of 72 minutes from a book on tape, continuous speech from a single male speaker, stored in 12 files averaging 6 minutes each. Speech rate was about 170 words/minute. Speech was sampled at 16 kHz and blocked into frames of 30 ms, spaced 10 ms apart. The first 8 MFCC coefficients [9] and their first difference [7] were used to form one 15 dimensional vector for each frame of speech data (the loudness coefficient $C0$ was used only for the difference coefficients).

3.2.2 Phonetic HMM models

All experiments to be described were done with left-to-right, continuous Gaussian density HMMs, using 25 mixture components per transition. 139 context-dependent phones were modeled, and full covariance matrices were pooled among allophones of a given phoneme.

3.2.3 Phonetic transcription graph

The text (orthography) of each utterance was converted into a phonetic graph in the following way: a graph building program used a pronunciation dictionary to translate each word into a word phonetic graph. The dictionary contains possible pronunciations for 60 000 words with an average of 2.2 pronunciations per word. Interword coarticulation effects were ignored.

Word graphs were then joined in their order of appearance in the utterance, with a "silence loop" between words, and (optional) breath sounds before and after each word. Figure 3.2 is an example of such a phonetic graph (except that initially the phonetic graph does not have times associated with its nodes).

3.2.4 HMM parameter estimation

The HMM model parameters are estimated by a procedure described in [7]: each transition in the models has be associated with a particular frame of data. In other words, the Viterbi paths through the internal states of the HMM models are needed. They get computed during point score calculations, but only their Viterbi score is actually kept in memory; once an exact segmentation is known, the large amount of memory needed to keep track of paths taken inside the individual phone models can be spared,

since the cost of recomputing them in a separate step constitutes only a small percentage of the total computation. This step uses the segmentation given by the block Viterbi algorithm to quickly align HMM models with data frames, and then the model parameters can be estimated.

3.3 Experimental results

3.3.1 Path convergence and look-ahead interval

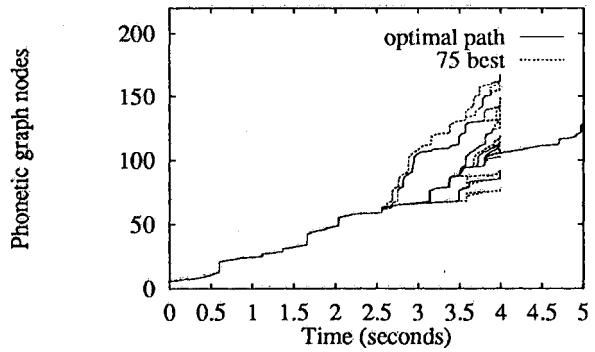

Fig. 3.3 Traces of 75 best scoring paths at window end.

Figure 3.3 is typical of what is observed when backtracking N best paths at the end of a window. In this example, B is 4 seconds and L is made equal to zero to allow backtracking over the entire interval from 0 to 4 seconds. The 75 best paths (at 4 seconds) merge in a single path in less than 1.50 seconds. Again the speech file used here had not been seen during the training. This suggests that the optimal path could merge with the best path in less than 1 second most of the time, i.e. that a look ahead interval B of a second or less could guarantee an optimal search.

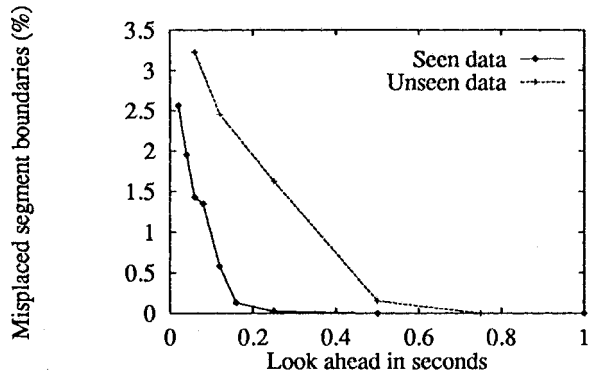

Fig. 3.4 Errors in segmentation as a function of look-ahead interval.

An experiment was run to determine exactly how large the look ahead interval B has to be for optimality. Figure 3.4 shows errors in segment boundary location as a function of the look-ahead interval, for speech data seen during the training or never seen before. The total of window length plus look-ahead interval was kept constant at 4 seconds, while look-ahead interval was varied from 0.06 to 2.0 seconds. Segmentations were compared to a reference segmentation obtained with a 3 seconds look-ahead interval. There were 3786 and 3873 segments in the seen and unseen speech data, respectively. The figure shows that using a look ahead interval of more than about 0.8 seconds is not necessary, even for new speech.

The window length L is made as large as permitted by computation and memory limits. In practice, we use a length of 3 seconds. The overhead for one second look-ahead then accounts for only one-fourth of total processor and memory use.

3.3.2 Computation and memory requirements

For a typical speech file of 6 minutes, the phonetic graph has about 14 000 nodes and is 36 000 frames long. Given a segmentation file from a previous iteration, the uncertainty window that limits nodes and times considered will be set to roughly 300 nodes by 600 frames (depending on its location in the trellis).

Over a total of 139 context-dependent phones, on average 70 will not appear in a particular window, and will not be considered (model pruning). Combination of entry time pruning and duration constraints will reduce remaining computation by about half. Total computation is in the order of 15 times real time on an HP Apollo 720 workstation when a 1 second look-ahead is used and uncertainty is plus or minus 2 seconds around segmentation times.

A window length of 3 seconds sets total memory use to around 10 Mbytes, with 3.3 Mbytes allocated for the trellis, most of the rest occupied by point scores computed in advance for each block.

4. CONCLUSION

Extending the sliding window idea, we developed a training algorithm requiring a fixed amount of memory, that can be used on unsegmented, unlimited length speech utterances. In our experiments selection of a single survivor path could be made optimal by looking at its future less than one second ahead. Retaining only one survivor at each window is particularly efficient when searching with block Viterbi - making available block Viterbi benefits such as entry and exit time pruning and duration constraints.

REFERENCES

[1] L.R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257-286, february 1989.

[2] K.F. Lee. "Context-Dependent Phonetic Hidden Markov Models for Speaker-Independent Continuous Speech Recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 38, no. 4, pp. 599-609, april 1990.

[3] A. Kriouile, J.F. Mari, and J.P. Haton. "L'algorithme VITERBI-BLOC pour la reconnaissance de la parole continue", XVIIIèmes Journées d'étude sur la Parole, pp. 207-211, 1990.

[4] R. Pieraccini, C. H. Lee, L.R. Rabiner. "Implementation aspects of large vocabulary recognition based on intraword and interword phonetic units," *Proceeding of DARPA Speech and Natural Language Workshop*, pp. 311-318, June 1990.

[5] P. Kenny, R. Hollan, V. Gupta, M. Lennig, P. Mermelstein, and D. O'Shaughnessy. "A*-admissible heuristics for rapid lexical access", *Proc. IEEE ICASSP*, pp. 689-692, 1991.

[6] E.A. Lee and D.G. Messerschmitt. *Digital Communication*, Boston: Kluwer Academic Publishers. 1988.

[7] L. Deng, P. Kenny, M. Lennig, V. Gupta, F. Seitz and P. Mermelstein. "Phonemic hidden Markov models with continuous mixture output densities for large vocabulary word recognition". *IEEE Transactions on Signal Processing*, vol. 39, pp. 1677-1681, 1991.

[8] J.N. Siddall. *Analytical decision-making in engineering design*, New Jersey: Prentice-Hall. 1972.

[9] S.B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," pp. 357-366, *IEEE Trans. Acoust., Speech and Signal Process.*, ASSP-28, 1980.