Phoneme segmentation and Voice activity detection

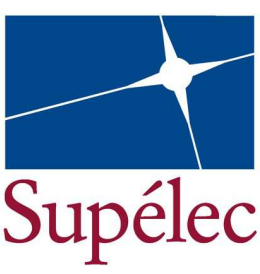
Joshua Winebarger

Report on Internship 2011

Phoneme segmentation and Voice activity detection

Joshua Winebarger
GeoStat team, INRIA Bordeaux-Sud Ouest

September 7, 2011



Contents

Preface 4

1. Phoneme Segmentation 5
   1. Introduction . . . 5
   2. Review of Existing TI Phoneme Segmentation Methods . . . 6
   3. Experimental Setup . . . 8
      3.1. Speech Corpus . . . 8
      3.2. Feature Extraction & Parameter Selection . . . 8
      3.3. Performance Measures . . . 9
   4. DISTBIC-SVF . . . 10
      4.1. Parameter Selection . . . 12
      4.2. Results . . . 13
   5. Segmentation by Growing Windows . . . 13
      5.1. OCD-Chen . . . 13
      5.2. Growing Windows . . . 13
   6. Segmentation by Divide and Conquer . . . 15
      6.1. DACDec2 . . . 15
      6.2. DACDec3 . . . 16
   7. Alternative Cepstral Features . . . 18
      7.1. Human Factor Cepstral Coefficients . . . 18
      7.2. Results with HFCCs . . . 19
      7.3. Impact of HFCCs on Performance . . . 20
   8. Summary & Comparison . . . 21
      8.1. Relative Performance of the Methods Tested . . . 21
      8.2. Work of [1] . . . 22
      8.3. Work of [2] . . . 23
      8.4. Comparison with several other works . . . 24
   9. Conclusion . . . 25

2. Voice Activity Detection 26
   1. Introduction . . . 26
   2. Discussion of the standards implemented . . . 27
   3. Experimental Setup . . . 29
      3.1. Test database . . . 29
      3.2. Performance measures . . . 30
   4. Results . . . 30
   5. Comparison to Other Works . . . 31
   6. Conclusion . . . 32

3. Outcome 34

A. Explanation of Methods and Development in Work 36
   1. Tools . . . 36
   2. Applying Greater Rigor in Phoneme Segmentation . . . 36
   3. Evaluation Paradigm for Phoneme Segmentation and Associated Work . . . 37
      3.1. TIMIT Database . . . 37
      3.2. Evaluating a Phoneme Segmentation Algorithm . . . 37
   4. Parallelization . . . 38
   5. Evaluation Paradigm for VAD and Associated Work . . . 38
      5.1. Adaptation of TIMIT for the Creation of a VAD Evaluation Database . . . 38
      5.2. Implementation of VAD Scoring . . . 39
      5.3. Evaluationg a given VAD . . . 39

B. Using ML Estimators in Maximum Log-Likelihood Ratios for ΔBIC 41
1. Description of Problem . . . 41
2. Case of univariate data . . . 42
3. Case of Multivariate Data . . . 43
4. Comparison to expression found in Delacourt, 2000 . . . 44
5. Case of diagonal covariance matrices . . . 45

C. Recursive mean and variance for OCD-Chen 46
1. OCD-Chen . . . 46
1.1. Computational Cost Analysis . . . 46
2. Recursive OCD-Chen . . . 47
2.1. Update of means . . . 47
2.2. Update of variances . . . 48
2.3. Practical Implementation . . . 49
2.4. Computational Cost Analysis . . . 49

D. Converting the Scores of Esposito 51

Preamble

This internship was intended to be a continuation of my work last year with the same team, whose focus is non-linear methods for complex signal analysis using concepts of scale invariance and particularly the development of a new multiscale microcanonical formalism (MMF). While the fields of application of this new formalism are diverse, one of them is speech processing. My contribution was exploratory research into innovative methods for text-independent phoneme segmentation which conform to a "linear" model, the goal being to provide a performance comparison with the "non-linear" MMF-based methods under development by the other team members.

This year I focused on two areas: a continuation of last year's work in phoneme segmentation, and implementation of voice activity detection algorithms. For the continuation of last year's work, I performed experiments with more rigor in order to better understand the results I obtained last year. I re-examined the algorithms I implemented last year and corrected discrepancies, and brought the implementations closer into line with standard practice. Some of the work to this end is described in a section in the Appendix A. I performed the requisite experiments to evaluate the performance of these methods on a standard database used for phoneme segmentation. I continued past this point with experiments on two other segmentation methods, in preparation for publication of a comprehensive journal paper. I made improvements to the functioning some of these methods, and in some instances I was able to improve the performance of the algorithms.

In addition to phoneme segmentation, the team is interested in applying the MMF to the field of Voice Activity Detection (VAD). It was desired that I implement several so-called "classical" VAD algorithms to serve as a basis for comparison for the new, non-linear algorithms which will be developed by the team in the future. As such I implemented four VAD algorithms commonly used as references in the literature to function as a standard reference for the new methods being developed. Further, I implemented a framework for evaluation of VAD algorithms. This consisted in devising methods for generating test databases for use in evaluating the performance of VAD algorithms and implementing them in code. Also under this effort, I wrote programs for scoring the output of these algorithms. I adapted existing code for two standard VADs to function within this framework, and finally evaluated these VADs under different conditions.

This report is divided as follows. Chapter 1 is devoted to phoneme segmentation. It describes the theory behind this field, an explanation of the functioning of the algorithms we tested, a description of our experimental setup, and our results testing on a standard database used for evaluating phoneme segmentation methods. We then compare our results to others in the literature and draw conclusions. Chapter 2 deals with VAD. We discuss the theory of VAD and its applications, followed by a brief overview of the four VAD methods tested. Our experimental setup is explained, followed by our experimental results. Finally Chapter 3 evaluates the internship and recommends areas for future work. A set of appendices follows, detailing the tools and methods used in our experiments, including the evaluation paradigm for our experiments. We also give detailed derivation of key a mathematical expression used in our work – the ΔBIC measure – and an exposition of an improvement we made on an important algorithm. Last, we provide an explanation relevant to the comparison of our results with those of a paper popular in the literature.

Special thanks go to my supervisor in my work in the GeoStat Team at INRIA, Dr. Khalid Daoudi, and co-supervisor, Dr. Hussein Yahia, for their guidance and wisdom. I would also like to thank Dr. Oriol Pont for his helpful advice and insight, as well as Vahid Khanagha for his providing a basis for the start of my work in speech processing and his knowledge in this area, both of the GeoStat team. Further my gratitude goes to Suman Maji and Dr. Harish Kumar for their encouragement, and Josy Baron for her assistance. Last, I thank Gilles Fleury, supervisor of the MATIS program at Suplec, for his help supervising my studies and preparation for this internship and his thoughts in reviewing it.

Chapter 1.

Phoneme Segmentation

Motivated by the similarities between speaker segmentation and phoneme segmentation, we examine in this paper the application of adaptations of three speaker segmentation detection methods to phoneme segmentation. Two of these methods incorporate the use of the spectral variation function, a method commonly used as a reference in phoneme segmentation tasks. We add further novelty through the use of Human Factor Cepstral Coefficients (HFCCs,) which are cepstral features using a relatively new alternate filter bank bandwidth scaling. By carrying out an evaluation of each method on the TIMIT database, a performance comparison between the different methods is provided. Our results show that the newly-introduced methods give performance comparable to other, very different text-independent methods found in the literature. We suggest that this latter finding supports a hypothesis found in another work which suggests a maximum has been reached in the performance of text-independent phoneme segmentation algorithms.

1. Introduction

Most of the developments in speech technology such as recognizers, synthesizers or coders strongly rely on corpus-based methodologies which require the availability of the precisely time-aligned labels of speech building units. The task of time-aligned labeling of the speech signal at the level of its smallest linguistic units (phonemes) is called phonetic segmentation. The most precise and accurate method for phonetic segmentation is by hand. However, manual segmentation of speech corpora is time-intensive and expensive. For this reason, automatic phonetic segmentation is of great importance and interest.

The most frequent approach is to adapt an HMM-based phonetic recognizer to the segmentation task by letting the recognizer know the phonetic or orthographic transcription and performing a forced alignment [3]. Such methods, referred to as "top-down" since they begin with linguistic assumptions trained on a database, provide a very high segmentation accuracy but impose linguistic constraints on the algorithm. Further they are restricted to the database used for training and thus cannot be applied to databases with different languages, contexts or accents. Such approaches which rely on an externally supplied transcription of the sentence for determining phoneme boundaries are called text-dependent (TD) segmentation methods.

The above-mentioned difficulties of text-dependent methods has caused the interest in the development of Text-independent methods which, though generally performing less well than TD methods, have the advantage of segmenting solely based on acoustic information of the speech signal. While not strictly language-independent, this approach lends itself more readily to multi-lingual applications. Since these methods do not rely on training from a speech corpus with its own particular acoustic properties, they may be more robust to changes in background noise [4]. Further, the lack of training is useful in absence of reliable phonetic transcriptions. Such methods are suitable for all those applications that may benefit from explicit speech segmentation, such as speech recognition, annotation, or coding. The proposed methods in this paper, belong to the TI class of segmentation methods.

We present in this paper, several methods for TI phoneme segmentation based on adaptations of speaker segmentation methods, or hybrids of speaker segmentation methods with SVF. Speaker segmentation is the task of automatically determining which speakers spoke when during an audio recording. The tasks of speaker and phoneme segmentation are similar, namely the detection of changepoints based on dissimilarity between segments of the speech signal. The main difference between the two methods is the scale involved. Whereas turns between speakers take place on the order of seconds, the length of phonemes is measured in milliseconds. However, the features extracted in both contexts are similar in scale, meaning that estimation of parameters from the features is more challenging in the case of phoneme segmentation due to fewer data points in the analysis windows.

Our first experiments with phoneme segmentation explored the hybrid of the spectral variation function (SVF) and DISTBIC. SVF is a popular phoneme segmentation method, which we will describe in more detail in the next section. Being a non-parametric method, it is computationally inexpensive but generally yields a high rate of false alarms (the insertion of erroneous transitions.) DISTBIC, on the other hand, was

introduced in [5] as a method for speaker segmentation (as referred to as speaker turn detection.) DISTBIC employs two stages, the second of which serves to eliminate erroneously inserted transitions. As will be seen, the high rate of detections of SVF works well with DISTBIC's error-reduction methods when SVF is used as a part of the first stage. We call this hybrid of DISTBIC and SVF "DISTBIC-SVF." Since DISTBIC, as we adapt it, is reasonably successful as a phoneme segmentation method, we hypothesized that other speaker segmentation methods might perform well.

Our goal is to show that speaker segmentation methods can be applied to phoneme segmentation and furthermore that this approach can be competitive with other state-of-the-art TI phoneme segmentation methods. To this end, this paper introduces additional methods from speaker segmentationbased on the divide-and-conquer (DACDec) paradigm. Like TI methods for phoneme segmentation, these speaker segmentation algorithms are unsupervised in that they employ only acoustic information.

As with the second stage of DISTBIC, the methods of DACDec perform a segmentation based on a distance measure originating from a statistical modeling framework. The assumption is that two adjacent sections of speech containing speech from the same speaker will have reasonably similar distributions, whereas two sections from different speakers will have dissimilar distributions, indicating a transition (a speaker turn) at the point between the two sections. The assumed distribution type is usually a multivariate Gaussian; the distance is then based on the dissimilarity between their means and variances or covariance matrices. We adapt this technique to detect not the transition between speakers, but between phonemes.

Further, we add novelty to our work by using Human Factor Cepstral Coefficients (HFCCs) as features for part of our experiments. HFCCs are a relatively new variant on the traditional Mel Frequency Cepstral Coefficients. Whereas with Mel Frequency Cepstral Coefficients (MFCCs) the front end filter bandwidth is tied to the filter center frequency, the filter bandwidth of HFCCs is decoupled from filter spacing to capitalize on the relationship between center frequency and critical bandwidth known from human psychoacoustics. We implement all four of our algorithms(SVF, DISTBIC-SVF, and the two DACDec algorithms) with MFCCs and HFCCs to provide a comparison between the results obtained using the two types of features.

The rest of the paper is organized as follows. In section 2 we review existing TI methods for phoneme segmentation. Section 3 describes our experimental setup including database, feature extraction, performance measures, and scoring. Sections 4 through 6 present the four speaker segmentation methods we use for phoneme segmentation. Results are given separately in each individual section. Each method uses the experimental setup of Section 3 with MFCCs as features. Procedures for parameter selection for each method are described in each section. Section 4 presents our previous work on the SVF and DISTBIC and its results. Section 6 describes the two Divide-and-Conquer methods and gives their results. In Section 7.1 we introduce the HFCCs and contrast them with MFCCs. Then we present results from all methods using the HFCCs as features. Section 8 summarizes our results and compares them to those obtained in other papers on TI phoneme segmentation. Last, Section 9 concludes the paper.

2. Review of Existing TI Phoneme Segmentation Methods

TI methods for phoneme segmentation tend to rely on distortion measures which quantify the spectral change between consecutive frames. Front end processing may differ by either applying a model such as linear predictive coding (model-based) or no model (model-free) before the calculation of the distortion measure. Afterwards, some method is used for translating this continuous distortion measure into a decision on the presence or absence of a phoneme transition.

One popular method is the Spectral Variation Function (SVF) which we implement in this paper. The SVF is a measure of the magnitude of overall spectral change from frame to frame, providing a way to quantify the quasi-instantaneous spectral change with a single value. Traditionally, it is computed as an angle between two normalized cepstral vectors separated by some frame distance, these vectors being the difference between the cepstrum and its average over a multi-frame window. One form of the SVF is found in [6], where its is the norm of the delta-cepstral coefficients for the frame k:

$$SVF_{\Delta cep}(k) = \sqrt{\sum_{m=1}^{p} [\Delta C_k(m)]^2} \tag{2.1}$$

with ΔC_k being the derivative of the cepstral coefficients for frame k and p being the number of coefficients. [6] uses this form of the SVF to give an upper bound on the number of phonemes in a segment. In our work we hypothesized that this tendency toward oversegmentation may provide useful candidate points for the second stage of DISTBIC, since it can only reject spurious segmentation candidates.

Applying the SVF to a signal produces a series or curve with peaks corresponding to areas of rapid, intense spectral change. Traditional approaches to finding a segmentation with this curve identify the phoneme transitions as the minima of its second derivative [7]. We followed this approach for our implementation of a segmentation with SVF. This gives a distortion measure which should take higher values at areas of rapid or intense spectral change. Afterwards, a number of procedures can be used to turn the series of SVF values into a decision on the location of the phonetic transitions, such as locating the maxima.

A similar method is the Delta Cepstal Function (DCF) which was introduced in [8]. There, it was compared with SVF as a method for constraining the transitions between phonemes in an HMM phone recognizer. In [1] it was used as a reference segmentation method. The DCF measure at a given frame t is computed in several steps. The difference between the cepstrum around time t is computed:

$$d_k(t) = C_k(t+1) - C_k(t-1), k = 1, \cdots, K \tag{2.2}$$

This measure d_k is then time-normalized over the duration of the signal:

$$\hat{c}_k(t) = d_k(t)/d_{k,max} \tag{2.3}$$

where

$$d_{k,max} = max_t|d_k(t)| \tag{2.4}$$

The sum over all cepstral coefficients is computed, followed by normalization:

$$c(t) = \hat{c}(t)/\hat{c}_{max} \tag{2.5}$$

where

$$\hat{c}_{max} = max_t\hat{c}(t) \tag{2.6}$$

The resulting value can be used to locate the phonetic transitions.

The Kullback-Leibler (KL) divergence is another spectral distance measure. [1] gives the following form which is applied directly to the spectral density:

$$KL = \int_{-\pi}^{\pi} K\left(\frac{S_1(\omega)}{S_2(\omega)}\right) d\omega \tag{2.7}$$

with $K(x) := x - \log x - 1$ and S_1 and S_2 being the spectral densities for two adjacent frames.

The KL divergence measure may also be applied to the distribution of the spectra or of features. When we consider this distribution to be gaussian, the KL divergence is easily calculated based on the means and standard deviations of the respective distributions.

Another model-free method recently developed by [4] operates strictly on the spectrum of of the speech signal. First, a pre-emphasis filter is applied to set formants at approximately equal amplitude levels. Windowing and FFT analysis are performed on very short windows with a small window shift. The authors motivate the choice of short length of the windows by the desire to capture the location of the main vocal tract excitation for voiced sounds. The FFT coefficients are compressed using a hyperbolic tangent function to simulate the non-linear sensitivity of human hearing. A cross-correlation matrix is computed from the frames of the entire signal, with the diagonal being the linear time axis. A special 2D filter composed of a square region and two triangular regions is slid along the diagonal. The result is a frame-by-frame spectral distortion curve. Since this curve may be noisy, it is passed through a minimax filter followed by peak masking. Then a local signal energy criterion is used to modulate the selection of the most significant peaks, which are chosen as the final detected transitions.

[1] mentions several methods and implements a few as reference, among them SVF, DCF, and KL. A selection of other methods mentioned or implemented includes the convex hull, Temporal Decomposition, multiscale segmentation, and wavelet transform. The authors introduce their own segmentation algorithm which is compatible with any multidimensional feature encoding scheme. In their work, they consider as encoding schemes MelBank features, MFCCs, and Log-area Ratios (the latter being introduced in [9].) The latter is an LPC-based technique based on the area ratio function of partial correlation coefficients drawn from the LPC model of the signal. The algorithm of [1] proceeds in several stages. Consider the encoded speech x. Let x_i be the speech encoded in feature coefficient i. We compute a change function J_i^a from x_i, using change-assessment window of length a. Peaks of this function are detected according to a relative threshold b. Then these peaks are stored in a matrix $S(i, n)$ where n is the feature frame number. Last, a fitting procedure combines groups of near-simultaneous transitions from the multiple coefficient series into unique indications of phoneme location.

Another work, [2], investigates the relationship between manually transcribed phoneme boundaries and the spectral transition measure (STM.) The algorithm is built around computing the STM based on the MFCC coefficients as follows:

$$STM(m) = \frac{1}{D}\left(\sum_{i=1}^{D} a_i^2(m)\right) \tag{2.8}$$

with D being the number of coefficients in the spectral vector and $a_i(m)$ the "regression coefficient" or rate of change of the MFCC:

$$a_i(m) = \frac{\sum_{n=-I}^{I} MFCC_i(n+m) * n}{\sum_{n=-I}^{I} n^2} \tag{2.9}$$

The maxims of this series of STM values is found, followed by the application of a set of heuristics to remove spurious peaks, similar to a method we will use in this paper.

3. Experimental Setup

3.1. Speech Corpus

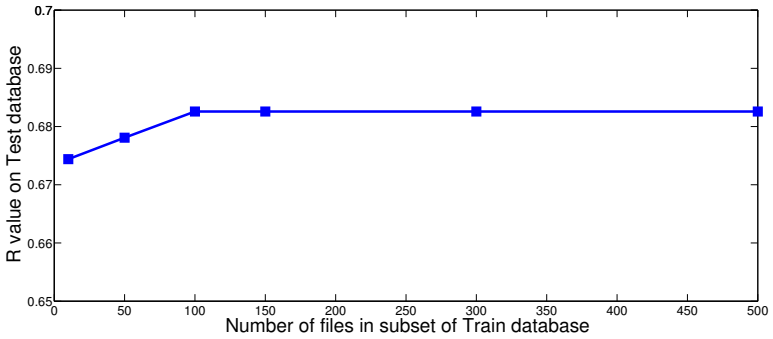
We wished to evaluate the quality of the segmentation produced by our algorithms in a way that is readily comparable to that of other algorithms. The evaluation metric of choice is usually a comparison to a manually annotated speech corpus. For this we used TIMIT, a corpus of read speech that is the most widely used for phoneme segmentation experiments [10]. Comprised of 6300 sentences spoken by 192 female and 438 male speakers drawn from 8 different American English dialects, the corpus is divided into a 'train' and a 'test' set. The sound files are sampled at 16 KHz. Each is accompanied by a text file containing a time-aligned phonetic annotation of the sentence spoken in the sound file. Every speaker provided 10 different sentence divided into three types: three phonetically diverse 'si' sentences, five phonetically compact 'sx' sentences which have the aim of providing good coverage of all American English phone pairs, and two 'sa' sentences used for evaluating dialect. As is customary in the literature, we excluded the latter sentences from our evaluations.

3.2. Feature Extraction & Parameter Selection

All experiments with the exception of those presented in section 7.1 used MFCCs as features. The first 12 coefficients (excluding the 0^{th} coefficient) were extracted, along with their 12 Δ-coefficients. The SVF segmentation used exclusively the Δ-coefficients, while other methods used the first 12 ordinary cepstral coefficients of which the Δ-coefficients are the derivative. Throughout our experiments, the MFCCs and their derivatives were computed using a modification of the *melcepst* function in the MATLAB toolbox Voicebox [11]. This modification was the use of the next power of 2 from the length of the frame. For instance, a 300-point frame would use a 512-point FFT. In any case the feature frame steps and lengths which were tried were powers or multiples of 2. All other settings besides feature frame length and step were set to default: a hamming window for the frames, triangular shaped filters in the mel domain, and filters acting in on absolute magnitude. We fixed the number of filters in the filter bank at 29 for correspondence with the same number of filters used in the features introduced in 7.1. MFCC frame and step size were varied in the range of 20ms and 10 ms, respectively as part of the parameter selection process and thus had no default value.

Our parameter selection was performed with the aim of maximizing R-value, an overall measure of segmentation quality which will be introduced in the next subsection. For parameter selection, we experimented on a subset of the *Train* portion of the TIMIT database which we call SmallTrain. Evaluation of each algorithm was carried out on both the full *Test* and *Train* portion. A key question in our studies was the optimum size of the *Train* subset used for tuning parameters. The larger the number of files in the subset, the more time would be required to tune files. On the other hand, in theory using too small a subset risked a choice of parameters giving poor performance on the larger database.
To solve the question of optimum training subset size, we ran a series of experiments using the SVF method. We randomly generated three databases of size N for $N = 10, 50, 100, 150, 300, 500$, for a total of 18 databases drawn from the *Train* part of TIMIT. We then tuned feature frame length and step of the SVF algorithm using MFCCs (SVF-MFCC) to find the pair giving the highest R value for each database. Tuning was done over 78 possible combinations of length and step. Each of these 18 configurations was then used in applying SVF-MFCC on the *Test* database, with the R value being noted. Finally, the R-value for each triplet of databases having the same N was averaged, yielding six R-values.

Figure 1.1.: R-value of SVF Trained on N-file subset of *Train* and tested on *Test*



We see that we reach a plateau at $N = 100$ files; for $N \geq 100$ the tuning algorithm selected the same configuration of feature frame length and step regardless of N or variation on the database. Based on this information, we chose to tune our algorithms on a randomly selected subset of 100 files from the *Train* part of TIMIT and test on the full *Train* and *Test*.

3.3. Performance Measures

Following the standard practice of the literature, we count a transition detected by the automatic segmentation algorithm as a hit if it falls with in a $\pm$20ms range of a manually transcribed transition (a reference transcription) [12]. If multiple automatically detected transitions fall within the the same range, we count only temporally-earliest one as a hit and the rest as insertions. If no automatically detected transition falls within the $\pm$20ms range, we count a miss.

Partial performance measures were used to score the performance of the algorithms on individual speech files and on the entire database: the Hit Rate (HR), which is the ratio of correctly detected boundaries to the number of reference transcriptions, the Oversegmentation (OS), which is the difference between the number of detected transitions and the number of reference transcriptions over the number of reference transcriptions, and the False Alarm Rate (FAR), which is the difference between the number of detections and hits over the number of detections. Here we treat each measure as a fraction between 0 and 1 rather than a percentage between 0 and 100.

$$HR = \frac{\text{hits}}{\text{transcriptions}} \times 100 \quad OS = \frac{\text{detections - transcriptions}}{\text{transcriptions}} \times 100 \quad FAR = \frac{\text{detections - hits}}{\text{detections}} \times 100$$

We further define the Precision (PCR) as $100 - FAR$, which is the likelihood that a given detected transition will be correct. HR on the other hand, measures likelihood that a reference transcription would be detected. Recall (RCL) is simply the fraction $\frac{HR}{100}$.

The composite performance measure the most often used in the literature is the $F1$ measure, a harmonic mean of PCR and HR:

$$F1 = \frac{2 \cdot PCR \cdot RCL}{PCR + RCL} \tag{3.1}$$

[13] showed that the $F1$ measure is lacking in sensitivity to "stochastic oversegmentation." This is to say that increases in $F1$ may be achieved with the insertion of boundaries at random locations unrelated to the speech signal. The authors of that study proposed a measure called the R-value which is not susceptible to this problem. The R-value is computed from two distances r_1 and r_2:

$$r_1 = \sqrt{(100 - HR)^2 + OS^2} \tag{3.2}$$

$$r_2 = \frac{-OS + HR - 100}{\sqrt{2}} \tag{3.3}$$

$$R = 1 - \frac{abs(r_1) + abs(r_2)}{200} \tag{3.4}$$

The first measure, r_1, is the euclidean distance between the achieved HR and OS and the ideal performance of $HR = 100$ and $OS = 0$. r_2 measures the distance between the achieved scores and a situation with no insertions.

4. DISTBIC-SVF

The DISTBIC algorithm was originally introduced by [5] for unsupervised speaker segmentation based on acoustic changes. [12] implemented DISTBIC as a phonemic segmentation method (albeit not by combining it with SVF.) In that work different model selection criteria besides BIC were tried. The authors combined these criteria with the replacement of the gaussian model with a generalized gamma distribution, resulting in very good performance when tested on the "Core" Test set of TIMIT, a subset of the Test set containing 24 speakers, two male and one female from each dialect region.

The algorithm beings by computing measures of quasi-instantaneous acoustic dissimilarity at a series of points covering the entire signal. The result is a distance curve with larger magnitudes indicating rapid changes in acoustic properties. Heuristics are then applied to select candidate transition points from this curve. Last, we verify these candidate transitions sequentially using a process of dynamic windowing with hypothesis testing.

Stage 1: SVF Distance Curve and Heuristics

As with most segmentation algorithms we begin with the extraction of multi-dimensional features which encode the acoustic information of the signal. The next step is the generation of a distance curve. For this step [5] uses a method called FixSlid-GLR, wherein two adjacent subwindows are slid over the signal and their dissimilarity is measured at each point with generalized likelihood ratio (GLR.) Here, we replace the GLR curve with an SVF curve. The motivation for doing so is that, were we to use FixSlid with GLR on the smaller scale of phoneme segmentation, the size of the fixed windows would be insufficient for reliably estimating the gaussians in the GLR. SVF, on the other hand, has been used previously for phoneme-scale measurements of acoustic dissimilarity.

First, we investigated SVF as a method for phoneme segmentation by itself. We used a simplified expression of the SVF [6], and identified phoneme transitions as the maxima of the SVF curve. We had only two free parameters for this algorithm: MFCC frame length and MFCC frame step size. We varied these two parameters in the range of 1/4 ms - 40 ms and 2 ms - 80 ms respectively. We found 10 ms step and 24 ms length gave the best R-value. These configurations are identical or close to that used by [1]. We give the best results of these experiments here in 1.1.

Table 1.1.: SVF optimal parameters and performance

	Optimal Parameters	
Features	Frame Length	Frame Step
MFCC	24 ms	10 ms

	TIMIT Test					**TIMIT Train**				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC	65.65	5.50	37.77	0.639	0.685	64.99	4.33	37.71	0.636	0.685

The first stage of DISTBIC builds on the information provided by a dissimilarity curve (the GLR curve in the work of [5], the SVF curve in our work) using set of heuristics examine the maxima of the distance curve and produce a set of candidate transition points. A series of rules are used to choose maxima which are more likely to correspond to phoneme transition points. It is recommended in [5] to low-pass filter the distance curve $d(t)$ prior to use of the heuristics. We do not include this step since the choice of the filter increases the space of parameters to be tuned, whereas our goal is to maintain simplicity in our system. We ensure a minimum distance between candidate peaks by enforcing a minimum distance h_2 between maxima. If two maxima are separated by less than this distance, they are merged by replacing them with a point located at their averaged position and possessing their averaged magnitude. To be chosen as a candidate, the difference between a maximum (max) and the minima to its left and right (min_l and min_r) must be greater than the standard deviation σ of the signal multiplied by tuning factor α:

$$\mathsf{d(max)} - \mathsf{d(min\_r)} > \alpha\sigma \text{ and } \mathsf{d(max)} - \mathsf{d(min\_l)} > \alpha\sigma$$

In the case of a maximum occurring at the left (resp. right) limit of the signal with no left (resp. right) minimum adjacent, we consider only the difference between the maximum and its right (resp. left) minimum.

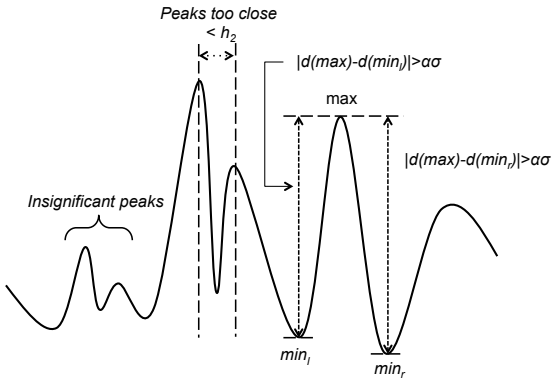


Figure 1.2.: Example of the heuristic measures involved in the heuristics of DISTBIC

Stage 2: Dynamic Windowing with ΔBIC

In the second stage of DISTBIC, which we refer to as DynWin, we verify the candidate transitions of the previous step using dynamic windowing with the ΔBIC criterion, which is a comparison of statistical models through the difference in their respective BIC model selection criteria. Consider a set of feature data $X = \{x_1, x_2, \cdots, x_{N_X}\} \in \mathbb{R}^d$ and various models $M = M_1, M_2, \cdots, M_k$. In order to select the model $\hat{M}$ best conforming to the data we choose to maximize the BIC criterion over M_i:

$$BIC(M_i, X) = \log p(X|\hat{\Theta}_i) - \lambda P \tag{4.1}$$

where $\hat{\Theta}_i$ is the maximum likelihood estimate of the parameters for model M_i, and λ is a tuning factor. P is a factor penalizing model complexity, being $P = \frac{m_i}{2} \log N_X$ where m_i is the number of model parameters in model M_i and N_X being the number of feature data points.

Next consider the feature vectors of two adjacent subwindows: $X = \{x_{j-N_X}, x_{j-N_X+1}, \cdot, x_j\}$ and $Y = \{x_j, x_{j+1}, \cdot, x_{j+N_Y}\}$ with their union being $Z = \{x_{j-N_X}, \cdots, x_{j+N_Y}\}$. We wish to know if the frame of feature j contains a transition point. We form two hypotheses:

- H_0: $Z \sim N(\mu_Z, \Sigma_Z)$ meaning that the two windows contain features from the same phoneme
- H_1: $X \sim N(\mu_X, \Sigma_X)$ and $Y \sim N(\mu_Y, \Sigma_Y)$ meaning that the windows contain two different phonemes

with μ and Σ being the mean vectors and covariance matrices of the windows.
To compare the two hypotheses we compute the ΔBIC score, which is the difference between the BIC scores for the models represented by H_0 and H_1:

$$\Delta BIC_{\{X,Y\}}(j) = BIC(H_1, X) - BIC(H_0, X) \tag{4.2}$$
$$= R(j) - \lambda P \tag{4.3}$$

where $R(j)$ is the log likelihood ratio given by:

$$R(j) = \frac{N_Z}{2} \log|\Sigma_Z| - \frac{N_X}{2} \log|\Sigma_X| - \frac{N_Y}{2} \log|\Sigma_Y| \tag{4.4}$$

with the penalty P depending on the model used. As we noted in the introduction, the time scale considered in our algorithms, relative to the duration of the features used, is relatively short when compared to the task of speaker segmentation. Thus, to reduce the number of statistical parameters to be estimated, we used only diagonal covariance matrices in all ΔBIC tests. Thus our penalty is $P = p \log(N_X + N_Y)$ where p is the dimension of the feature vectors. Here we have $m = \frac{1}{2}2p$ since there are p parameters for the mean vector and p parameters for the diagonal of the covariance matrix. With $\Delta\text{BIC} = 0$ we have no preference for either hypothesis. With $\Delta\text{BIC} > 0$ H_1 is stronger than H_0 and in this case a transition can be inferred.

Perhaps unique to our implementation, again extract features from the underlying signal for the second stage, independent of the features used to compute the first stage. This is to say we recompute the MFCCs (or HFCCs in later sections) using feature frame lengths and step sizes that may differ than those used in the first stage. We justify this difference in features between the two stages of the algorithm by noting the smaller time scale of phoneme segmentation, to which we made reference in the introduction. The second stage of dynamic windowing performs statistical tests using windows defined by the candidate transitions

of the first stage. Because, ideally, these candidate transitions should be interspaced at the same scale as the time between phonemes in natural speech, it is probable that statistical windows will contain only a handful of feature data (when using feature step size and length appropriate for the first stage of the algorithm.) There is thus the risk of poor estimation of the means and variances involved in the statistical tests, with a corresponding degradation in performance. We therefore find it appropriate to decouple the feature frame length and step size of both stages, since the second stage may require much smaller frame step and length. Indeed, this will prove to be the case with the Divide-and-Conquer algorithm.

The dynamic windowing algorithm uses this ΔBIC hypothesis test as follows. Define the set of P candidate transitions found using the first stage:

$$\hat{T}_{svf} = \{\hat{t}^0, \hat{t}^1, \cdots, \hat{t}^P\} \tag{4.5}$$

We form three windows of feature data (which we extracted anew from the signal in this, the second stage of the algorithm,) called X, Y, and Z. X contains those feature data between the limits $[\hat{t}^0, \hat{t}]$. Y is adjacent to X and contains those feature data in the range $[\hat{t}^1, \hat{t}^2]$. Z spans both X and Y, containing the feature data in the limits $[\hat{t}^0, \hat{t}^2]$. Thus, we have two adjacent windows, X and Y, the bounds of each being a set of three adjacent candidate transitions. The window Z spans the windows X and Y.

We compute the ΔBIC score for the combination of windows X, Y, and Z. If $\Delta\text{BIC} > 0$, we shift all three windows at their default size, with X being $[\hat{t}^1, \hat{t}^2]$, Y being $[\hat{t}^2, \hat{t}^3]$, and Z being $[\hat{t}^1, \hat{t}^3]$. On the other hand, if the candidate is rejected, we expand the right-hand limit of X by one candidate transition. That is to say, X is now $[\hat{t}^0, \hat{t}^2]$. We shift Y accordingly Y by one candidate transition, such that its bounds are $[\hat{t}^2, \hat{t}^3]$. Z again spans both X and Y. Then we perform again a ΔBIC test. The process is repeated until we have covered the entire signal. Finally, the candidates that were verified are considered the output of the algorithm. The process is repeated until we have covered the entire signal. Those candidates that were verified are considered the output of the DISTBIC algorithm.

The operation of the DISTBIC-SVF algorithm is illustrated in 1.5a. The top panel shows the ground truth or position of the manual reference transcriptions as dashed lines which continue to the scoring stage. The next panel below shows the distance curve with significant peaks as selected by the heuristics as diamonds. These form the candidates of SVF-Heur. Next white (resp. black) rectangles indicate the left (resp. right) subwindows of the dynamic windowing stage, shown progressing from the top of the panel to the bottom. The last two lines show the resulting detections and their scoring against the ground truth with misses, false alarms, and correct detections indicated.

4.1. Parameter Selection

The DISTBIC-SVF algorithm has the following free parameters:

1. MFCC frame length for SVF-Heur
2. MFCC frame step for SVF-Heur
3. α
4. h_2
5. MFCC frame length for DynWin
6. MFCC frame step for DynWin
7. λ.

Due to the fact that we must select feature frame step and length for both stages of the algorithm independently, we are faced with a potentially huge number of possible parameter combinations. To save time and computational resources, we constrained the combination of parameters by choosing them in two steps. We evaluated the first stage as its own segmentation method, producing a set of candidate parameters for the first stage. Then the full algorithm was tested using the candidates for the first stage and a full range of parameters for the second stage.

With the knowledge that DynWin could only eliminate mistaken transitions, we preferred increased HR to decreased FAR for the SVF-Heur stage. We performed trials of SVF-Heur with 160 different configurations of the parameters α, MFCC frame step, and MFCC frame length. Among those which produced segmentations with HR $> 75\%$, we selected the 20 combinations of parameters giving the highest ratio of HR to OS and the 20 combinations of parameters giving the highest ratio of HR to FAR for a total of 40 candidates.

These 40 candidates were used in the next step of tuning the DISTBIC algorithm as a means to speed the tuning. Here our parameter space included a large possibility of different λ, MFCC frame step, and MFCC frame length for the DynWin stage, with possible parameters for the SVF-Heur stage being drawn from

the 40 candidates. We used a first sub-step wherein the tuning database was a 25-file subset of the 100-file subset of TIMIT Train. From these experiments we chose the 10% of parameter configurations giving the best R value. These candidates configurations were tried on the 100-file subset, with the candidate giving the best R value being selected as the final configuration. This final configuration was tested on the full TIMIT Test database and the full TIMIT Train database.

4.2. Results

Here we present our results from testing DISTBIC-SVF. Optimal parameters are given in table 1.2. For conservation of space we have abbreviated the Frame Length of the first stage (SVF-Heur) to "FL1", the Frame Step to "FS1", and likewise for the second stage (DynWin.)

Table 1.2.: DISTBIC-SVF optimal parameters and performance

	SVF-Heur			**Dywin**		
Features	FL1	FS1	α	FL2	FS2	λ
MFCC	20 ms	6 ms	0.4	24 ms	6 ms	0.2

	TIMIT Test					**TIMIT Train**				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC	72.87	-0.2941	26.911	0.7298	0.7694	72.79	-0.6209	26.75	0.7302	0.7700

5. Segmentation by Growing Windows

In segmentation by growing windows, termed 'WinGrow' in [14] we start with a small analysis window placed at the beginning of the speech signal. For each analysis window, we perform a process of one-change-point-detection (OCD-Chen) which detects a candidate changepoint. If the maximum ΔBIC score is greater than zero, we mark the transition at that point and move the window. If not, we increase the range of the analysis window until a transition is found.

5.1. OCD-Chen

One-change-point, introduced by [15], seeks to detect a single changepoint in the analysis window Z of length N_Z. The window is divided into two adjacent subwindows X_i and Y_i where i is the feature frame index within Z locate at the interface between the two subwindows. i (and thus the relative lengths of X_i and Y_i is incremented between a minim value i_{min} and maximum value $N_Z - i_{min}$. For each i we compute $\Delta BIC_{X_i,Y_i}(i)$. The result is a ΔBIC curve. If the maximum value of the latter so computed is greater than 0, we mark the time index of the maximum as a changepoint, otherwise we assume there is no changepoint in the signal.
[14] state that we impose a minimum and maximum i in order to ensure a length for the windows X_i and Y_i which is sufficient to reliably estimate the covariance matrices from which the ΔBIC statistic is calculated. The authors recommend a value in the range of 30 to 50.

OCD-Chen, in a straightforward implementation, incurs a high computational cost. This is especially the case when performed on very large window (such as those the length of the entire signal as with DACDec2.) In Appendix C we develop a means for recursively performing this algorithm, with significant time savings as evidenced in our analysis of the computational cost in the same section.

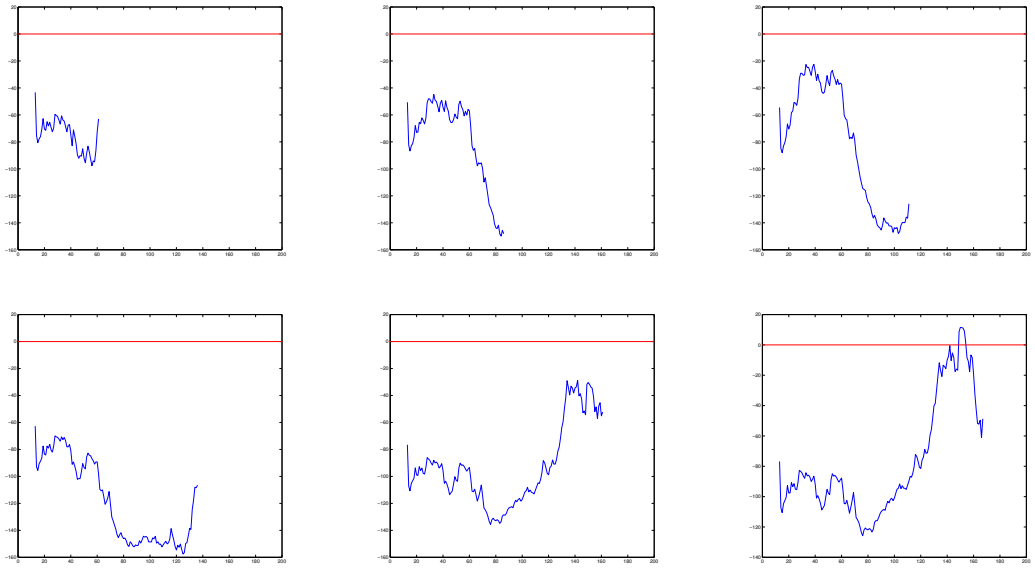
5.2. Growing Windows

To apply OCD-Chen in a method to detect multiple changepoints in the signal, we embed it in a framework of dynamically expanding windows which search the signal. The algorithm begins with an initial analysis window of length $N_{win-min}$ wherein we perform OCD-Chen. We repeatedly grow the analysis window in length by $N_win - inc$ samples and perform OCD-Chen until a changepoint is detected. If a changepoint is detected we mark a transition, revert the analysis window to its initial size and place its starting point at the transition point. This process continues until the entire signal has been covered. In our implementation, we

perform this pass forwards, starting from the beginning of the signal, and backwards, starting from the end of the signal. We did this because whether a phoneme boundary in the signal is identified or not may depend on the length of the analysis window, which in turn depends on whether it was preceded by a detection.

Each subfigure in figure 1.3 shows a snapshot of the progression the WinGrow algorithm. Snapshots are taken every fifty steps in the extension of the window, starting from the beginning of the speech signal, with the window growing by one signal point every step. In each figure is the ΔBIC curve output by OCD-Chen for the given analysis window. As the growing analysis window expands, the shape of the curve changes, until finally in the last subfigure a point in the curve exceeds 0, resulting in the detection of a transition.

Figure 1.3.: Progression of ΔBIC curves with expanding windows. The scale does not change from subfigure to subfigure.



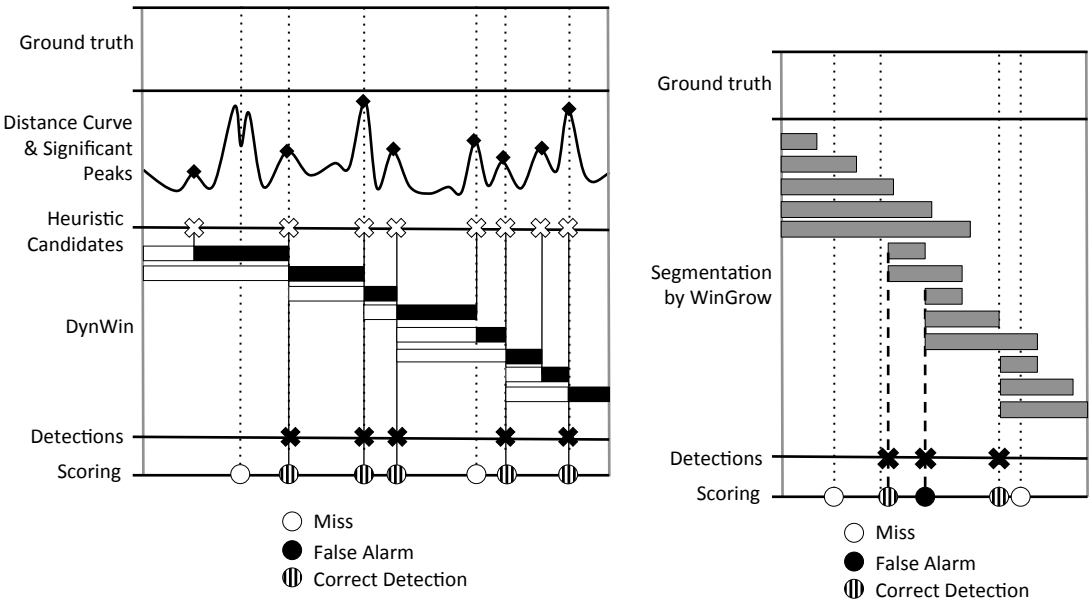
We can infer in looking at the pre-final subfigure that had the value of λ been lower, a transition would have been detected at the large, earlier peak.

Figure 1.5b shows a conceptual example of the operation of the WinGrow algorithm. In the top panel the ground truth reference transcription is shown with dotted lines that extend all the way down to the scoring stage. In th next panel the shaded bars indicate the OCD-Chen windows which grow. When a transition is detected a dashed black line is dropped to show the detection as an 'X'. Misses, false alarms, and correct detections are indicated. We show here the last reference transcription being missed due to insufficient window length caused by the detection preceding it. This type of error is systematic in WinGrow [14] and motivates the work of section 6.

Because WinGrow repeatedly scans the signal using OCD-Chen, the time required for running this algorithm on the smallest of databases is quite high. We see in section 1.1 that the time taken to perform OCD-Chen on a window of length N is $O(N^2)$. Assuming a window growth step of 1, to grow a window to size M (assuming no transitions are detected) requires $O((M(M+1)/2 \times M^2))$ time. [14] estimate in their paper that to detect k transition points using WinGrow requires $O(km^3)$ where m is the maximum growing window size (after being reached the analysis window slides rather than grows.) Even with the economizations we develop for OCD-Chen in Appendix C, the computational cost proved to be too prohibitive for running experiments.

WinGrow also suffers from a systematic error which hinders its performance wherein false alarms induce missed detections. Namely, if it produces a false detection just prior to the location of a true transition, the true transition may go undetected, due to the fact that the false alarm will cause the detection process to restart. In that case, the window size will be reset with its left boundary being the false detection. Now, when OCD-Chen scans the new analysis window, the true transition may be located too close to the left boundary to produce a $\Delta BIC > 0$ at the true transition.

Figure 1.4.: Conceptual illustration of DISTBIC and WinGrow operation.



(a) DISTBIC

(b) WinGrow

6. Segmentation by Divide and Conquer

As noted before, in addition to systematic error, WinGrow incurs a high computational due to its repeated scanning of the same segments of speech in expanding the analysis window. [14] proposes three algorithms aiming to be more efficient and avoid some of the systematic errors of WinGrow. The second being an improvement on the first and the third being based on a somewhat different principle than the first two, we chose to implement only the second and third algorithms. The Divide and Conquer algorithms proceed in two stages: a Divide stage, and a Combine stage, both recursive.

6.1. DACDec2

Unlike WinGrow which starts with a small analysis window at the beginning of the signal, DACDec2 starts its Divide stage with an analysis window covering the whole signal. Let us denote this window as a "parent" window. OCD-Chen is applied to find a single division point at time $\hat{t_i}$ corresponding to the maximum ΔBIC value produced. The signal is divided at $\hat{t_1}$ resulting in two "child" subwindows W_1^i and W_2^i. These children are considered parents and are each divided through the same process, producing their own child nodes. This process is repeated recursively. The algorithm is prohibited from partitioning any window whose length is less than N_min. When no segment in the signal can be divided further, the Divide stage stops. The candidate points resulting from the Divide stage are simply the ensemble of the nodes, which form the boundaries between those subwindows having no children of their own.

Next we proceed to the Combine stage, where we recursively verify only those nodes at which the ΔBIC was negative (those nodes with $\Delta BIC > 0$ are automatically labeled as detected transitions.) Verification of a division point is performed with a ΔBIC test as in Dynamic Windowing, however the order of verifying the division points is different. Verification starts with the temporally earliest division point, node j. This point will be verified with the its two neighboring segments X and Y defined as those parts of the signal between the division point of node j and its two neighboring division points. If $\Delta BIC_{X,Y} > 0$ segments X and Y are preserved and the division point is preserved. Otherwise, we erase the division point and merge segments X and Y. In either event, we consider node j in the tree resolved. We next seek to verify j', the parent of node j. However, if node j' has other unresolved child nodes, those nodes must be resolved before resolving node j'. In this recursive fashion we work our way towards verifying the division point corresponding to the initial node 1.

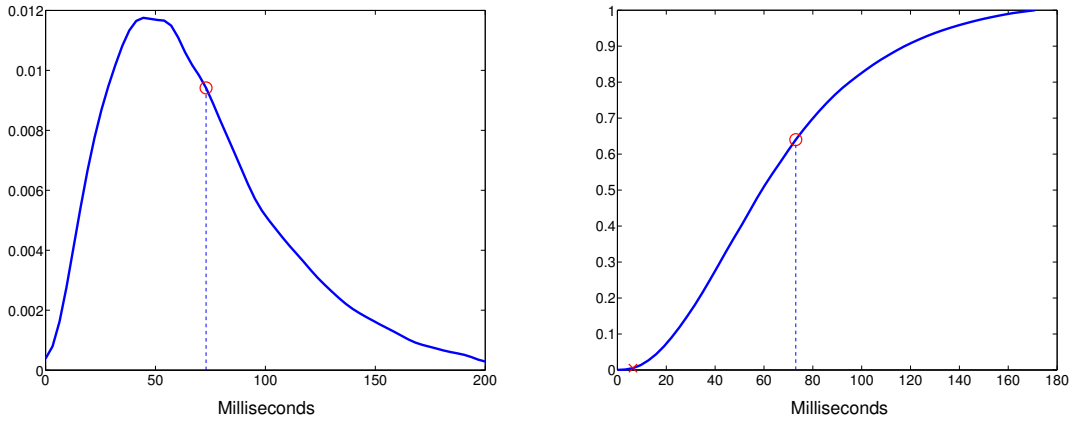
Parameter Selection

Selection of parameters for DACDec2 was comparatively simple. There were only four parameters of interest:

1. MFCC frame length
2. MFCC frame step
3. N_{min}
4. λ

where N_{min} is the minimum allowable window duration between transitions. We wished to ground this minimum duration in the speech data of TIMIT, in that we sought a duration shorter than which only a few phonemes in the database would be. We examined both the Train and Test sets of TIMIT for information on the distribution of phoneme lengths in the database. The results are shown in subfigures 1.6a and 1.6b. The mean phoneme length was around 73 ms. Only 1% of all phonemes were shorter than 8ms, indicated by the red cross in the lower left hand corner of the CDF figure. For this reason, we set the minimum allowable distance to 8 ms.

Figure 1.5.: Density and CDF of phoneme lengths in TIMIT. Mean indicated as red circle.



(a) PDF

(b) CDF

The three free parameters, λ, MFCC frame length and step size, were tuned over a range of values. The search for these parameters was effected through tuning over a subset of the TIMIT database.

Results

Results for the best performance of DACDec2-OCDChen in our experiments are presented in table 1.3.

Table 1.3.: DACDec2-OCDChen optimal parameters and performance

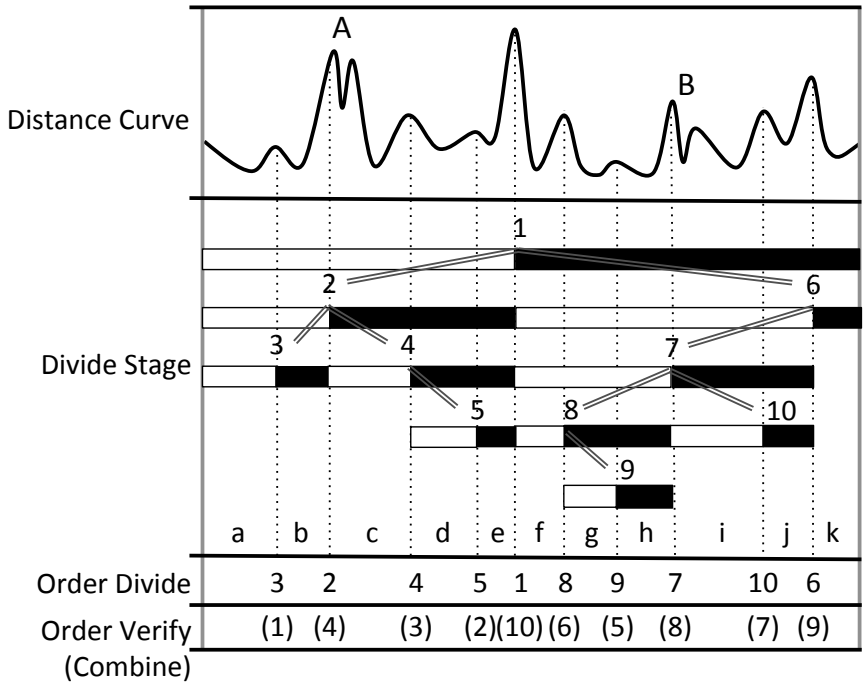
Features	FL	FS	λ	N_{min}
MFCC	2 ms	0.25 ms	1.8	8 ms

	TIMIT Test					**TIMIT Train**				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC	70.93	-5.544	24.90	0.7296	0.7689	71.00	-6.030	24.44	0.7321	0.7707

6.2. DACDec3

DACDec3 is a hybrid algorithm with commonalities with DISTBIC and DACDec2. The algorithm can be conceptually divided into three stages. In the first stage, we generate candidate transitions by applying heuristics to a distance curve. Next in the Divide stage we organize these candidates into a hierarchy base on their associate distance value. Last, in the Combine stage, we use the hierarchy to recursively verify each node in the hierarchy with a ΔBIC test. Thus the algorithm replaces the recursive OCD-Chen of

Figure 1.6.: Illustration of DACDec3 operation



DACDec2 with candidate transition generation similar to DISTBIC, but verifies these candidates similarly to DACDec2.

For the first stage, [14] uses FixSlid with the GLR distortion measure to produce this curve. As with DISTBIC-SVF, we replace FixSlid with SVF in our work. Then we apply the same heuristics found in Section 4 to perform SVF-Heur. The result is a set of candidate points having time indices $\hat{T}_{svf} = \{\hat{t}^1_{svf}, \cdots, \hat{t}^P_{svf}\}$ with associated SVF magnitudes $V_{svf} = \{v^1_{svf}, \cdots, v^P_{svf}\}$.

A modified Divide stage is then applied. The changepoints found by this Divide stage are the same as those of $\hat{T}_{svf}$. In effect, the purpose of this divide stage is to establish a parent-child node hierarchy for these candidate transitions based on the GLR value at each $\hat{t}$.
As in DACDec2 we start with a window $W = [t_{start}, t_{fin}]$ covering the whole signal. We search for the index $\hat{i} = \arg\max_{i \in I}\{v^i_{svf}\}$ with I being the set of indices of those $\hat{t}_{svf}$ inside the window: $t_{start} < \hat{t}^i_{svf} < t_{fin}$. This gives the maximum value in V_{svf} in the range of W. W is then partitioned at point $\hat{t}^{\hat{i}}$ yielding two child windows. The preceding steps are applied to recursively partition the signal. All the while, the division of any window smaller than a minimum length is prevented. Since windows may only be divided at candidate transition points, this entails that some candidate transitions may not pass the Divide stage. When no further division is possible, the Divide algorithm stops.

Afterwards, the combine stage precedes almost identically to that of DACDec2, except that as with Dyn-Win in DISTBIC, we perform a second round of feature extraction with the possibility of feature frame lengths and steps different than those used in SVF-Heur. The changepoints detected by the SVF-Heur, having been placed in a node hierarchy by the Divide Stage, are verified recursively using ΔBIC tests. The key difference is that since we have GLR values for each node rather than ΔBIC values, we verify all the nodes.

Figure 1.6 provides an illustration of the process. The Distance Curve panel shows an example SVF curve. The Divide Stage panel shows, from top to bottom, the progressive partitioning of the signal with the white and black (left and right, respectively) windows. Numbers appear at each partition point to indicate conceptual nodes, with double lines linking each node. The temporal order of the division is indicated below, along with the order of verification of each node in the Combine stage. Points A and B on the distance curve indicate two otherwise significant peaks which were not translated into divide points due to the minimum window length constraint. We can use this figure to elucidate the Combine stage in which we verify recursively each node. Consider Divide Node 4, which is Combine Node 3. Now before verifying this

node, its child, Node 2, must be verified. Node 2 is invariably verified with a ΔBIC test on the segments d and e. If these segments confirm the node, the corresponding transition (the boundary between segments d and e) will be preserved as a detection of the algorithm. Then Node 3 will be verified with segment c on the left and d on the right. If on the other hand the ΔBIC test rejects Node 2, d and e will be merged. In that case, Node 3 will be verified with segments c and $\{d, e\}$.

Parameter Selection

Since DACDec3 has much in common with DISTBIC-SVF, it is natural that the parameter selection proceeded similarly. The parameter selection for the two parametrized stages proceeded separately for reasons of computational cost. Searches for the best-performing parameters were performed on a subset of the TIMIT database. With the assumption that the purpose of the Divide and Combine stages is solely to eliminate mistaken transitions, we tuned the SVF-Heur stage separately with view towards oversegmentation. More specifically, when choosing parameters for the SVF-Heur stage we preferred increased HR to decreased FAR. As noted in 4.1, the parameter selection for this stage done in common for DISTBIC-SVF and DACDec3. Parameters in question for this stage were:

1. MFCC frame length for SVF-Heur
2. MFCC frame step for SVF-Heur
3. α
4. h_2

The Divide stage of DACDec3 involves no parameters but rather the simple application of rules.
As noted, we performed a second round of feature extraction for the Combine stage. The parameters tuned were:

1. MFCC frame length for Combine stage
2. MFCC frame step for Combine stage
3. λ

Results

Table 1.4.: DACDec3-SVF optimal parameters and performance

	SVF-Heur			**DACDec**		
Feature Config.	FL1	FS1	α	FL2	FS2	λ
MFCC-MFCC	10 ms	6 ms	0.4	2 ms	0.125 ms	1.8

	TIMIT Test					**TIMIT Train**				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC-MFCC	72.69	-2.287	25.60	0.7353	0.7745	73.15	-2.286	25.14	0.7399	0.7784

7. Alternative Cepstral Features

7.1. Human Factor Cepstral Coefficients

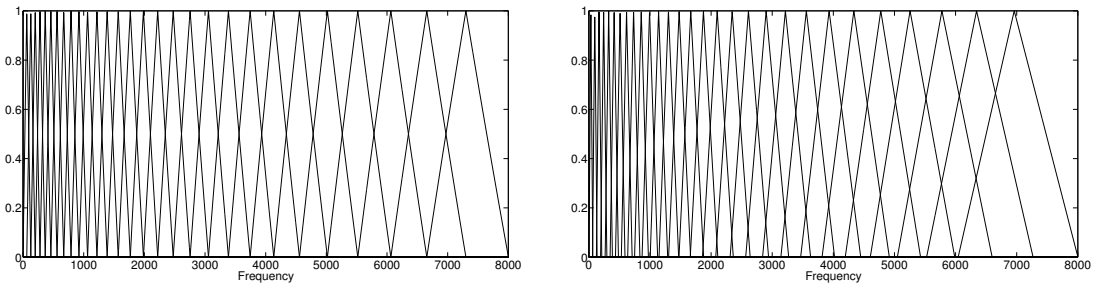
Mel Frequency Cepstal Coefficients (MFCC) have been a standard speech feature for a generation. In extracting MFCCs, a short time window of speech undergoes a Fourier Transform, after which it is passed through a Mel-scaled filter bank of triangular overlapping filters. Next, the log is taken of the sum of the magnitude coefficients output by each filter. This is followed by the application of a discrete cosine transform of the mel log magnitudes. The result is a set of coefficients.
MFCCs have several advantages, including greater robustness to noise than linear prediction coefficients [16]. The center frequencies of the mel filter are perceptually spaced, which is to say equally spaced in mel frequency. However, the endpoints of each filters is determined by the center frequency of the adjacent filter, a design choice motivated more by convenience than by fidelity to the human auditory system. Further, when the number of filters in the mel filter bank is changed, the filter bandwidth is changed.

Human Factor Cepstral Coefficients (HFCC) were introduced in by Skowronski and Harris [17] as an update of the MFCC framework with the aim of rectifying this latter shortcomings of MFCCs. In HFCCs,

the filter bandwidth is decoupled from the filter spacing. Instead the filter width is set according to by the relationship between center frequency and critical bandwidth of the human auditory system. [16]. Specifically, bandwidth is determined from equivalent rectangular bandwidth (ERB) proposed by Moore and Glasberg [18]:

$$ERB = 6.23f_c^2 + 93.39f_c + 28.52 \tag{7.1}$$

where the f_c corresponds to center frequency in kHz. In 2004, the Skowronski and Harris introduced a further refinement, HFCC-E, in which the above bandwidth is scaled by an ERB scale or E-factor. The authors showed that this linear scaling increased the noise robustness in their tests of an automatic speech recognition system.



(a) MFCC (b) HFCC

Figure 1.7.: The filter banks used in our experiments

HFCCs have previously been used as part of a language-dependent phoneme segmentation evaluation in the work of [19]. In that work, the segmentation of several context-dependent and context-dependent HMMs, each using a different configuration of speech features, were fused to produce a better result than those of any one method. In doing so a comparison of the performance of various features was provided. Seven feature sets were tried:

1. Human Factor Cepstral Coefficients (HFCC-E)
2. Linear Frequency Cepstral Coefficients (LFCC)
3. Mel-Frequency Cepstral Coefficients (MFCC)
4. Perceptual Linear Prediction (PLP)
5. Wavelet-Packet Features (WPF)
6. Subband-based Cepstral Parameters (SBC)
7. Mixed Wavelet Packet Advanced Combinational Encoder (MWP-ACE)

While different HMM and feature configurations worked best for different classes of phoneme transitions, HFCCs gave the best overall performance.

We investigated the effects of replacing MFCCs with HFCCs in each of our algorithms. While TIMIT speech is clean and thus our algorithms are unlikely to benefit from noise robustness, the move to a more perceptually-motivated filter arrangement was seen as a possible benefit. In our experiments we generated HFCC features using the code available from the website of one of the authors [20]. We used the default settings for all configuration parameters, including a 512-point FFT (as with the MFCCs), and 29 filters in the filter bank, in accordance with the number used in [16]. Two exceptions to these defaults were the feature frame length and step, which were tuned as part of our parameter selection.

The tables in the following subsection provide a summary of the scores produced with the replacement of MFCCs by HFCCs using default settings. Note that multi-stage methods, such as DACDec3, were tested only with HFCCs for either the second or both stages.

7.2. Results with HFCCs

We provide here the optimal parameters found for the use of MFCCs and HFCCs in our algorithms and the resulting performance on the TIMIT Test and Train databases. The DISTBIC-SVF and DACDec3-SVF algorithms use two stages. DISTBIC-SVF uses SVF and Heuristics for the first stage and dynamic windowing for the second, and DACDec3-SVF uses SVF and heuristics for the first and Divide-and-Conquer

for the second. These two stages may be performed with either feature type, however we settled on three configurations:

I. MFCC-MFCC: MFCCs for both stages

II. HFCC-MFCC: HFCCs for the first stage, MFCCs for the second

III. HFCC-HFCC: HFCCs for both stages

Table 1.5.: SVF optimal parameters and performance

Features	Frame Length	Frame Step
MFCC	24 ms	10 ms
HFCC	32 ms	10 ms

	TIMIT Test					**TIMIT Train**				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC	65.65	5.50	37.77	0.639	0.685	64.99	4.33	37.71	0.636	0.685
HFCC	67.70	11.04	39.03	0.642	0.676	67.27	9.07	38.32	0.644	0.682

Table 1.6.: DISTBIC-SVF optimal parameters and performance

	SVF-Heur			**DynWin**		
Feature Config.	FL1	FS1	α	FL2	FS2	λ
MFCC-MFCC	20 ms	6 ms	0.4	24 ms	6 ms	0.2
HFCC-MFCC	24 ms	6 ms	0.4	10 ms	6 ms	0.2
HFCC-HFCC	20 ms	6 ms	0.5	8 ms	6 ms	0.2

	TIMIT Test					**TIMIT Train**				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC-MFCC	72.87	-0.2941	26.911	0.7298	0.7694	72.79	-0.6209	26.75	0.7302	0.7700
HFCC-MFCC	71.40	0.7450	29.13	0.7114	0.7532	71.44	-0.7333	28.03	0.7170	0.7588
HFCC-HFCC	69.79	-6.84	25.09	0.7226	0.7625	69.77	-8.001	24.16	0.7268	0.7650

Table 1.7.: DACDec2-OCDChen optimal parameters and performance

Features	FL	FS	λ	N_{min}
MFCC	2 ms	0.25 ms	1.8	8 ms
HFCC	4 ms	0.25 ms	2.8	8 ms

	TIMIT Test					**TIMIT Train**				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC	70.93	-5.544	24.90	0.7296	0.7689	71.00	-6.030	24.44	0.7321	0.7707
HFCC	71.70	-1.465	27.23	0.7223	0.7634	71.79	-1.859	26.85	0.7246	0.7655

7.3. Impact of HFCCs on Performance

We expected the decoupling of filter bandwidth from center frequency provided by the HFCCs to yield increased performance, since the latter is a more faithful implementation of the processing of the human auditory system. Further, the work of [16], wherein the HFCCs were used in a speech recognition experiment on the TI-46 digits database, showed promising performance for this new feature. Here, the decreased filter bandwidth associated with increasing numbers of filters in MFCCs was not a problem as we kept the

Table 1.8.: DACDec3-SVF optimal parameters and performance

	SVF-Heur			DACDec		
Feature Config.	FL1	FS1	α	FL2	FS2	λ
MFCC-MFCC	10 ms	6 ms	0.4	2 ms	0.125 ms	1.8
HFCC-MFCC	20 ms	6 ms	0.4	2 ms	0.125 ms	1.6
HFCC-HFCC	16 ms	6 ms	0.4	4.5 ms	0.25 ms	2

	TIMIT Test					TIMIT Train				
Features	HR	OS	FA	**F1**	**R**	HR	OS	FA	**F1**	**R**
MFCC-MFCC	72.69	-2.287	25.60	0.7353	0.7745	73.15	-2.286	25.14	0.7399	0.7784
HFCC-MFCC	73.20	-4.071	23.69	0.7472	0.7841	73.22	-5.121	22.83	0.7514	0.7871
HFCC-HFCC	72.70	-3.707	24.51	0.7407	0.7788	72.97	-4.264	23.78	0.746	0.7827

number of filters constant. Furthermore we used the same number of filters in our implementation of the MFCCs and in our implementation of HFCCs, thus the only difference between the two was the bandwidth of the filters.
The scores yielded from our experiments on TIMIT show a mixed picture of what performance increases are possible using HFCCs rather than MFCCs. Some methods, such as SVF or DACDec2-OCDChen, showed slightly diminished performance using HFCCs rather than MFCCs. DISTBIC-SVF showed the best performance using solely MFCCs for both stages, with solely HFCCs for both stages giving second-best performance and a mix of MFCCs and HFCCs giving the lowest performance. In that case, HFCCs did not improve on MFCCs either. Only with DACDec3-SVF was a performance increase seen using HFCCs. There the best performance was achieved using HFCCs for the first, with MFCCs being used for the second stage. The next best performance was seen using HFCCs for both stages. For all cases it was rare that the difference in performance resulting from using the two different features amounted to more than 1 % in terms of F1 or R-value, raising the possibility that the differences seen were due only to chance. On the other hand, the fact that scores from tests on the two databases (Test with over a thousand files and Train with over four thousand files) both showed this mild relative weakness for HFCCs probably points to some subtle disadvantage of the HFCCs, at least in this test and in this implementation. It is also possible that our use of a different database, TIMIT, than [16], contributed to the non-appearance of performance improvements.

8. Summary & Comparison

8.1. Relative Performance of the Methods Tested

A common theme was that performance on the TIMIT Train set was usually slightly better than performance on the Test set, and in any case never inferior to the latter. As noted in the prior section, the effect of HFCCs was difficult to distinguish from chance, and perhaps even a negative one. All of the methods derived from speaker segmentation techniques performed significantly better than the SVF, by up to 10 percentage points at maximum in terms of R Value. Oversegmentation was reduced, and the False Alarm rate was reduced by 10 % or more for all such methods as compared to SVF. The main source of performance gains in the SVF-based methods, DISTBIC-SVF and DACDec3, was the ability to oversegment the signal first using SVF, then eliminate spurious segmentations using Heuristics and ΔBIC. This lead to a higher Hit Rate with a lower False Alarm rate. DACDec2, operating on a different principle, achieved comparable performance. In fact its F1 and R-value were nearly identical to those of DISTBIC-SVF when using MFCCs, the difference being a lower Hit Rate and a lower False Alarm Rate. Ultimately DACDec3-SVF achieved the greatest performance of all methods, improving on DISTBIC-SVF by roughly 1 - 1.5 %.

One welcome outcome was similarity of values yielding optimal performance. The two methods sharing SVF-Heur – DISTBIC-SVF and DACDec3-SVF – shared roughly the same values for optima feature frame length and step as well as α for that step in their computations. Similarly, the two methods using forms of Divide and Conquer – DACDec2-OCDChen and DACDec3-SVF – required fairly similar values of frame length and step as well as λ for this latter stage. These two similarities were not planned – all algorithms were tuned over a wide but reasonable range of values for the aforementioned free parameters.
On the other hand, the difference in optimal feature frame length, frame step, and λ between DynWin and

DACDec was surprising. Whereas dynamic windowing found its optimal values with anywhere between 24 and 8 ms length and 6 ms step – not too far from the standard 20 ms length and 10 ms step commonly used in the literature– Divide and Conquer required very small frame lengths and very short steps. One possible reason for why this was so may have been the use of the ΔBIC test, which relies on the estimation of means and variances. If one is attempting to use the hypothesis test based on two temporally short windows, it may be difficult to have enough samples in the two windows to reliably estimate their variances. A smaller frame size and step would provide more features and thus more samples. On the other hand, if frame length is two small, there may be insufficient information encoded in any given frame to reliably model the underlying speech signal. The optimum frame length and step is surely a compromise between at least these two factors in algorithms using ΔBIC hypothesis tests. The question remains, however, as to why we did not see a similar small frame length and step as optimal for DynWin, since it is a method using ΔBIC tests as well. Further experimentation would be necessary to identify the root of this divergence.

In order to provide some context for our findings, we next examine the results of some other works in the literature.

8.2. Work of [1]

[1] is a frequently-cited source for Text-Independent segmentation, and thus we felt it indispensable to provide a comparison. However, it was found that [1] uses a measure of False Alarms that takes into account the number of frames in the signal, which is not habitual in the literature. This measure of False Alarms is thus dependent on frame size, and improves with smaller frames, an implementation which is not only unusual but probably undesirable as well, since ideally, performance measures should be agnostic with regard to the encoding of the signal.
A naive use of this FA score would lead to $F1$ and R-Values which do not reflect the same assumptions as other works. Based on the information provided by the authors including the number of frames in their databases, their methodology for choosing the best-scoring algorithm, and their methods for computing the partial performance measures, we translated their results into scores which reflect the habitual assumptions and methods presented in section 3.3. For a detailed accounting of how the scores of [1] were translated, see Appendix D.

Here we give a comparison between our results with the standard method of SVF having used this translation of scores.

Table 1.9.: SVF Scores reported here and [1]

Method	**HR**	**OS**	**FA**	**F1**	**R**
Our SVF-MFCC	65.65	5.50	37.77	0.639	0.685
Our SVF-HFCC	67.70	11.04	39.03	0.642	0.676
Esposito SVF-MFCC	67	3.2	35	0.66	0.71

We see that our scores are in the same range, with the implementation of [1] having a small but significant advantage over ours. This general agreement should re-assure us of the correctness of our implementation of this standard method.

Next, we compare our methods with the method developed by the authors of [1]. These resulting translated scores are presented in table 1.10.

Table 1.10.: Best performance of the algorithm from [1] as translated from the information therein

Coding Scheme	**(a, b, c)**	**HR**	**OS**	**FA**	**F1**	**R**
8-melbank	(2,optimal,5)	82	18.32	30.69	0.75	0.743
5-MFCC		76	12.31	31.33	0.72	0.736
Log Area Ratio		70	6.31	34.16	0.68	0.72

The closest analog with our methods was the 5-MFCC encoding, which used 13 coefficients as compared with our 12-coefficients. In that case, our best-performing method under MFCC encoding, DACDec3-SVF, gave a 1.5% improvement in terms of F1 and an approximately 4% improvement in terms of R value. If

we loosen the restriction on using the same encoding scheme and compare our best method with the best method of [1] irrespective of features used, we would compare the 8-melbank algorithm to DACDec3-SVF using a mix of HFCCs and MFCCs. In that case, our method gave a 2.7% improvement in F1 and an approximately 4% improvement in R-value. It should be noted that we tuned our algorithms for R-value rather than F1, and in our experience there is some trade-off between the two measures, since R-value prizes a low oversegmentation more than F1.

We note also an advantage of our methods over those of [1]. The scores given in table 1.10 are those associated with ROC curves developed in Section 4.3 of that paper. These ROC curves resulted from the progressive incrementation of one of the free parameters for the authors' algorithms, observing the effects on their measures of false alarm rate P_{fa} and miss rate $P_m = 1 - P_c$. (We ignore the difficulties associated with this false alarm measure, discussed above and in much greater detail in Appendix D.) For the generation of these ROC curves, the authors fix two of their algorithm's three free parameters a and c to 2 and 5, while varying b. While it is not clear by what procedure and using what database a and c came to be fixed at these values (the paper talks of an "external mechanism" for this purpose in its section on the selection of a, b, and c), we know from the paper that the authors used the full TIMIT Test database for the creation of these curves, necessitating the testing of their system for different values of b over their test database. Thus, effectively, at least one of their parameters has been tuned on the evaluation database, which is not proper practice. On the other hand, our algorithms were trained using a small subset (100 files) of the Train database, then evaluated on both the Test and Train databases. Thus, thus the performance gains we achieved become more significant because our methods did not necessitate tuning on a large number of files, much less tuning on the evaluation database itself.

8.3. Work of [2]

Another work of note is that of [2]. We reviewed the method briefly in section 2. The authors report the following statistics for the testing of their algorithm on the Train set of TIMIT:

Table 1.11.: Results reported in [2]

	Total Manual	Detected Auto.	Missed Auto.	Inserted Auto.
Count	172 460	145 950	26 510	48 566

The authors state that " approximately 85 % of the manually placed phone boundaries from the training part of TIMIT were detected by the automatic method ...", leading us to assume that the column header "detected" is the equivalent of our measure of hits. The distribution of time deviation between these 145 950 hits and their manual reference transcriptions is given in Figures 3 and 4 of the paper, with deviations of up to 100 ms. This indicates that the authors counted any automatically generated segmentation as a hit if it fell within 100 ms of a reference segmentation, a very wide margin. Traditionally in phoneme segmentation, we are interested in only those automatic segmentations falling within 20 ms of a reference segmentation. The cumulative histogram in Figure 4 of the paper shows that 90 % of all automatically generated segmentations falling within 100 ms of a reference transcription fell within 20 ms of that transcription. Thus in effect we have hits $= 0.9 \times 145950 = 131355$ with the rest of the originally defined "hits" being in fact insertions: insertions $= 145950 \times 0.1 + 48566 = 63161$ Using these values we can compute the following scores (not found in the paper of [2]):

Table 1.12.: Scores derived from results reported in [2]

HR	OS	FA	**F1**	**R**
76.17	12.79	32.47	0.7159	0.7353

The false alarm rate is a bit higher than that of most of our methods to that of many of our methods, and the oversegmentation rate is quite a bit higher. The F1 and R values seem to be driven by the higher hit rate. This higher hit rate is not as it seems however. The authors state that "in order to perform a the comparison between the manually and automatically placed phone boundaries the former are converted to the closest adjacent frame positions." Recall that we stated in section 3.3 that in scoring an automatically generated phoneme segmentation against a manual reference transcription, it is standard practice to count an automatically generated boundary as a hit only if it falls within 20 ms of a reference transcription. Note

also that the authors use a frame step size of 10 ms. The authors' practice of converting the manually transcribed transitions to the closest frame position has the effect of increasing the range in which an automatic transition is valid. Consider as an illustration the example of an automatically generated transition placed at 40 ms (the fourth frame,) along with a reference transcription existing at 17 ms. Whereas normally the automatic transition would be counted as an insertion because $40 - 17 > 20$, by the method of [2], the reference transcription will be brought to 20 ms, and thus within the 20 ms range of the automatic transition, resulting in a hit. Now consider if the reference transcription were found at 23 ms. It would be converted to 20 ms, still resulting in a hit. Or consider the case of a reference transcription existing at 14 ms. It will be converted to 10 ms and thus further away from the 20 ms range around the automatic transition, but even without conversion it would not have resulted in a hit. Since any transition halfway between the center of two frames (such as 15 ms) will be rounded up, we have effectively increased the hit zone around automatic transitions to 25 ms. The larger countable area will result in more hits and fewer misses and insertions, explaining the increased hit rate and F1. And because the total number of automatically generated transitions will remain the same, so will the oversegmentation rate (as hinted at by the relatively large value of this measure.) Thus the results in this paper are not directly comparable to those in our paper. Instead, given the understanding that the standard meaning of the 20 ms margin corresponds to a 15 ms margin in their work, we can use linear interpolation of the normalized histogram in the authors' paper to attempt an estimate of the percent of automatically generated segments falling within what the authors consider to be a 15 % margin: $\frac{0.7+0.9}{2} = 0.8$. Using this we have the following equivalence with the standard measures: Have made this attempt to adjust for the unusual effects of the authors' matching each manual segmentation

Table 1.13.: Scores derived from results reported in [2]

HR	OS	FA	**F1**	**R**
67.70	12.79	39.97	0.6363	0.6669

to the frame step size, we can see that had this not been done, their method would probably have yielded scores quite similar to our own SVF-MFCC and SVF-HFCC, even without heuristics.

8.4. Comparison with several other works

We mentioned in section 2 the work of [4] which introduced a novel Text Independent segmentation method based on non-linear filtering. That paper tested their algorithm on the TIMIT Test set as well as a Finnish in-house corpus. We note that they investigated the parameter dependency of their algorithm, a process akin to tuning, using a 200 file subset of the TIMIT Test database, whereas our algorithms were tuned on a smaller portion of the TIMIT Train database. [4] gives the results for several other "blind" segmentation (Text-Independent) algorithms reported by other authors. We reproduce that table, including the results from our three best performing algorithms, along with the results of [2].

Table 1.14.: Comparison of the results of several TI segmentation methods when testing on TIMIT

Algorithm	HR	OS	**F1**	**R**
Our DACDec3-SVF-MFCC-HFCC	73.2	-4.07	0.75	0.78
Our DISTBIC-SVF-MFCC-MFCC	72.9	-0.294	0.73	0.77
Our DACDec2-MFCC	70.9	-5.54	0.73	0.77
Khanagha et. al MMF-ACC (2011)[21]	72.4	6.42	0.70	0.74
Khanagha et. al MMF-LLRT (2011) [21]	72.6	1.64	0.72	0.76
Räsänen et. al (2009) [4]	71.9	-6.90	0.76	0.78
Aversano et al. (2001) [22]	73.58	0.00	0.74	0.77
Esposito & Aversano (2005)[1]	82	18.32	0.75	0.743
Estevan et al. (2007)	76.00	0.00	0.76	0.80

[4] perform a study on the segmentation accuracy of transitions for all pairs of phoneme types (nasals to fricatives, lax vowels to glides, etc.) and finds that while some pairs are well detected on average, others are not. Supporting this they find the distribution of the deviations of automatically generated segmentations from manual segmentations to support the notion that the transition between certain phoneme types are systematically under-detected. This is also supported by an examination of F0 and energy contours for

certain words. There, many types of transitions are too gradual or almost non-existent to be detected by the spectral change methods employed in "bottom-up" – that is to say Text-Independent – methods, whereas the occasional spectral splitting of certain phones into two or more "subphone" will cause a tendency towards unwanted insertions. Along with the fact that the methods cited yield approximately the same performance despite differing in their specific algorithmic approaches, this tends to point to a maximum possible performance having been reached with text-independent methods for phoneme segmentation.
Our methods introduced in this paper and earlier works are based on techniques used for speaker segmentation. The fact that they reach approximately the same performance as other methods supports the hypothesis that the maximum possible performance from TI phoneme segmentation is around an R value of 0.78 to 0.80, which in any case is less than what is possible with text-dependent methods.

9. Conclusion

We gave a discussion of the theory and practice associated with text-independent phoneme segmentation, including some existing algorithms. We introduced three new distinct phoneme segmentation algorithms based on methods for speaker segmentation. Two of these methods, DISTBIC and DACDec3 we hybridized with a classical nonparametric measure used in phoneme segmentation, SVF, to produce new variants thereupon. In implementing one algorithm, DACDec2, we improved on the standard implementation of OCD-Chen by creating a recursive version, thereby greatly reducing the computational cost associated with this procedure. The TIMIT Test and Train databases were used in experiments wherein we tuned our algorithms on a small number of files from the Train set and tested them on both databases. The resulting performance greatly outperformed standard SVF and was comparable to that achieved by the algorithms of other authors. In our earlier remarks on the difference between speaker and phoneme segmentation, we noted that the time scales of statistical analysis windows are significantly shorter in the latter, which implies that the statistical tests in phoneme segmentation algorithms will be forced to estimate parameters using a smaller number of data. In this light, the fact that we achieved performance comparable to other state-of-the-art phoneme segmentation algorithms using algorithms adapted from speaker segmentation becomes more impressive.

Comparing the performance resulting from the use of an alternative cepstral filter bank (HFCC features) to that yielded by standard features (MFCCs), we found a slight diminishment in performance using the new features. While we speculated that this may be due to our use of a different database than the authors originating these features, more detailed experiments are needed to investigate why the HFCCs did not contribute to enhanced performance as expected.

It was noted that another author put forward the notion that a performance limit has been reached in Text-Independent phoneme segmentation, namely that such algorithms are likely unable to exceed a performance of 78 - 80 % R-value (at least on the TIMIT database, being the database considered.) Our algorithms, despite being based on a different framework than others considered in the literature, and each being distinct from the other, gave maximum performances in this range. We find therefore that our experiments provide supporting evidence to the aforementioned notion. This is perhaps the greatest contribution of our work.

Chapter 2.

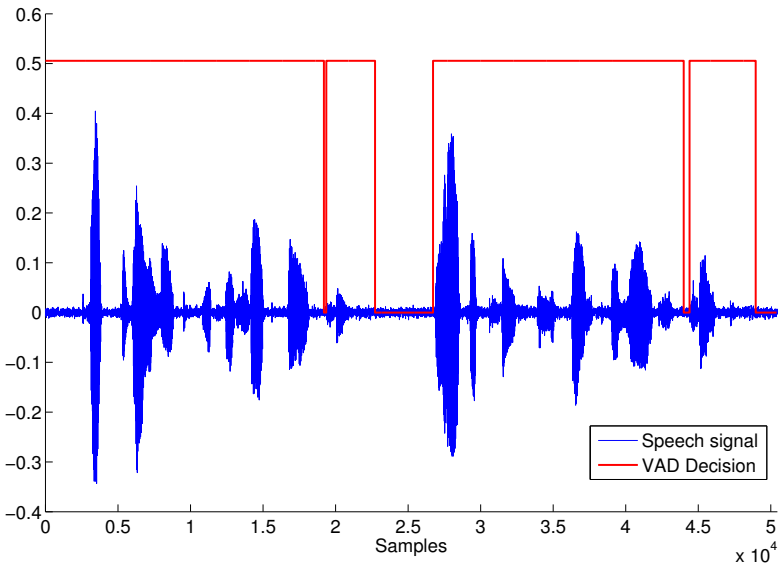
Voice Activity Detection

Voice Activity Detection (VAD) is a common component of several types of speech systems, including speech recognizers and speech coders. Improvements in VAD have a substantial impact on the quality of these systems. These improvements are typically evaluated through testing on corpuses of speech. Over the last decade, a few methods for VAD have come to be considered as standard and are widely used in the literature. The use of these methods in the literature facilitates comparison between new VAD algorithms, even in the absence of a common corpus for VAD testing, since authors' algorithms can be judged relative to their improvements over these standards. We implemented four of these standards and performed experiments on the TIMIT database, the results of which are given in the following section.

1. Introduction

VAD concerns itself with the automatic distinction between speech and noise or silence in sound recordings and live environments. Given, for example, an audio recording a VAD system should make a binary decision of speech/non-speech for all samples in the signal or at least on a frame-by-frame basis. The result is a label for the recording indicating which time segments contain speech and which do not. A VAD system's utility is partly measured by its robustness to noise. VAD is a key component of speech recognition systems, especially in noisy environments. If a VAD is applied to a noisy speech signal prior to its being input to a speech recognizer, the frames judged to contain only noise may be "dropped" from the input to the recognizer, thereby reducing the rate at which falsely inserted words are produced. VAD is also used for low-rate speech encoding, wherein a higher-quality speech coder may be applied to encode only those parts of the signal containing speech, while in decoding those non-speech portions of the signal may be merely simulated by generating comfort noise. Last, VAD is useful for enhancing the quality of speech. If portions of the signal containing only noise can be identified, the spectrum of the noise may be estimated. This information may then be used to produce a cleaner signal.

Figure 2.1.: Illustration of VAD



Based on the what seems to be customary in the literature, we sought to implement the following four VAD standards our work as a reference:

- G.729B published by the ITU

- AMR1 and AMR2 published by ETSI
- The Statistical VAD of Sohn

We tested these VAD algorithms on the TIMIT database, traditionally used for phoneme segmentation under various configurations of speakers with white gaussian noise at several signal to noise ratios (SNR). The following sections provide a brief discussion of the workings of each method, followed by a description of our experimental setup. Finally, we give the results of our experiments.

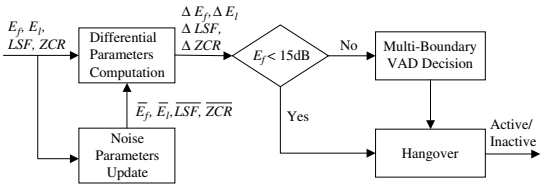
2. Discussion of the standards implemented

G.729 is a speech codec (an algorithm for encoding and decoding speech) for the purposes of efficient speech data compression and transmission, published and maintained by the International Telecommunications Union (ITU.) Associated with G.729 are a number of annexes, amongst them G.729B, which provides for discontinuous transmission through the use of a VAD to enable high-bitrate coding only during speech and not during noise. The algorithm makes a VAD decision for 20 ms frames spaced at 10 ms. The decision is based on four parameters within each frame:

- Full band energy difference: $\Delta E_f = \bar{E}_f - E_f$
- Low-band energy difference: $\Delta E_l = \bar{E}_l - E_l$
- Spectral distortion: $\Delta LSF = \sum_{i=0}^{9} (L\bar{S}F_i - LSF_i)^2$
- Zero-crossing rate difference: $\Delta ZC = \bar{ZC} - ZC$

where LSF_i is the i^{th} line spectral frequency and ZC is the zero-crossing rate, a measure of the number of times the signal changes sign. $\bar{E}_f$, $\bar{E}_l$, $L\bar{S}F_i$, and $\bar{ZC}$ are parameters (full-band energy, low-band energy, line spectral frequencies, and zero-crossing rate) characterizing the noise part of the signal, which are updated during non-speech regions. The decision is made based on region classification in this four-dimensional space, followed by a "hangover" technique, which extends delays the transition between speech and non-speech in the VAD decision. [23] A block diagram of the G.729B algorithm is provided in figure 2.2.

Figure 2.2.: Block diagram of the G.729B VAD algorithm as seen in [23]



The hangover technique merits greater discussion. It is done because the energy is often low near the end of segments of speech. Thus, frames preceded by frames having been declared as speech are more easily labeled as speech in an effort to reduce the premature labeling of speech as non-speech and to reduce the "jitter", or high-frequency transition between speech and non-speech. A hangover mechanism may take the form of a counter that is activated upon the treatment of a frame, which would otherwise be labeled as non-speech but was preceded by a speech frame. This counter may be enabled or disabled if certain conditions are met, such as the energy of the frame being below a certain threshold. Other more sophisticated strategies such as hidden markov models (HMM) are also employed in the literature.

Another standards organization, ETSI, produces two different Adaptive Multi-Rate (AMR) VADs, AMR1 and AMR2 for use in its speech coder. The AMR1 breaks the signal into nine non-uniform frequency subbands using filter banks, wherein the lower frequency bands have smaller bandwidths. For each frame the subband energy is calculated followed by an estimate of the SNR for each subband. This estimate requires a calculation of the noise energy, which is accomplished through a first-order autoregressive (AR) model. The speech-nonspeech decision is produced by comparing the sum of the subband SNRs with an adaptive threshold. A hangover scheme is implemented. [23] A block diagram is given in figure 2.3.

The AMR2 algorithm also decomposes the signal into subbands, but using the Fast Fourier Transform (FFT) instead of a filter bank. The energy of each of the 16 non-uniform subbands is computed. We also compute the SNR for each subband using the signal and the noise spectra. During speech, the background noise energy is adapted using an autoregressive model. The speech-nonspeech decision is made on the basis of the SNR estimate compared against an adaptive threshold. This threshold increases for highly fluctuating

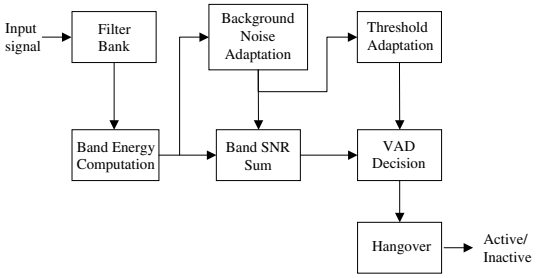


Figure 2.3.: Block diagram of the AMR1 VAD algorithm as seen in [23]

SNR, which is to say that the variance of the SNR over an ensemble of frames is used to modulate the the threshold. Additionally, the hangover delay reduced with increasing peak-to-average SNR. This average SNR is computed using an AR-adaptation of the instantaneous SNR. [23] A block diagram is given in figure 2.4.

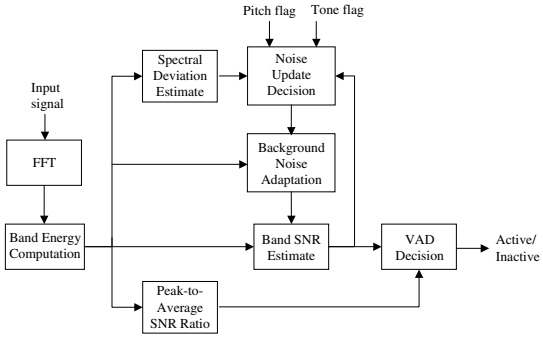


Figure 2.4.: Block diagram of the AMR2 VAD algorithm as seen in [23]

The above-discussed methods are based on rules and various measures of the signal energy and spectrum. [24] introduced a different style of algorithm which relies on a statistical model of the signal. This is the VAD of Sohn. We begin with the following approach. Basing the speech-nonspeech decision on the feature vectors of the signal x, we normally assume the noise and speech are additive, and consider two hypotheses:

$$H_0 : x = n \tag{2.1}$$

$$H_1 : x = n + s \tag{2.2}$$

The optimal decision rule that minimizes the error probability is a Bayes classifier. Given the vector x, we choose H_0 or H_1 depending on which has the largest posterior probability.

$$P(H_1|x) \underset{H_0}{\overset{H_1}{\gtrless}} P(H_0|x) \tag{2.3}$$

which based on Bayes' rule resumes to:

$$\frac{P(x|H_1)}{P(x|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \frac{P(H_0)}{P(H_1)} \tag{2.4}$$

for which the right hand side can simply be called η, a threshold, yielding

$$\frac{P(x|H_1)}{P(x|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \eta \tag{2.5}$$

Assuming that the data, being discrete fourier transform (DFT) coefficient vectors of dimension L are asymptotically independent gaussian random variables, we have for the hypothesis of noise:

$$p(X|H_0) = \sum_{k=0}^{L-1} \frac{1}{\pi\lambda_N(k)} \exp\left\{-\frac{|X_k|^2}{\lambda_N(k)}\right\} \tag{2.6}$$

and for the hypothesis of speech plus noise:

$$p(X|H_1) = \sum_{k=0}^{L-1} \frac{1}{\pi[\lambda_N(k) + \lambda_S(k)]} \exp\left\{-\frac{|X_k|^2}{\lambda_N(k) + \lambda_S(k)}\right\} \tag{2.7}$$

with $\lambda_N(k)$ and $\lambda_S(k)$ being the variances of the noise and speech in the k^{th} DFT subband. The expression in 2.5 then becomes:

$$\Lambda_k = \frac{p(X_k|H_1)}{p(X_k|H_0)} = \frac{1}{1+\xi_k} \exp\left\{\frac{\gamma_k \xi_k}{1+\xi_k}\right\} \tag{2.8}$$

where $\xi_k \triangleq \lambda_S(k)/\lambda_N(k)$ and $\gamma_k \triangleq |X_k|^2/\lambda_N(k)$ are the a priori and a posteriori SNRs.

The decision rule is then:

$$\log\Lambda = \frac{1}{L}\sum_{k=0}^{L-1} \log\Lambda_k \underset{H_0}{\overset{H_1}{\gtrless}} \eta \tag{2.9}$$

$\lambda_N(k)$ is assumed known, leaving to be estimated ξ_k. These unknown parameters are estimated using the maximum likelihood criterion. The authors employ a hangover scheme based on HMMs.

3. Experimental Setup

3.1. Test database

There exist different ways to evaluate the performance of a VAD. In applications where the VAD is a component of a larger system, such as a speech recognizer or a speech coder, one examines the difference in performance resulting from incorporating or leaving out the VAD from the system in question. For evaluating a VAD by itself, the common choice is to compare its output to that of a hand-labeled database. Many different, proprietary databases are used in the literature, making direct comparison of results difficult. Some authors, such as [25] use the phoneme-segmentation database TIMIT. This can be useful since TIMIT is a large, widely-used database, and the time indices indicating the However, the pitfalls lie in the fact that the files in the database contain only one sentence of speech per file, usually with no interruptions in the sentence. Evaluating a VAD on these types of files is uninteresting because there is little opportunity for the decision to label one section of the signal as speech to affect another large stretch of the signal. The logical solution is to concatenate sentences, but this opens questions as to how many and with what spacing and amount of non-speech, as well as whether all sentences in the concatenation should come from the same speaker. We discuss the details of our solution to these questions in Appendix section 5.1 As a comparison, [25] states that the authors use concatenations of two sentences, leaving unanswered most of the above questions.

For our experiments we created four databases of 100 concatenations. Each concatenation contained four TIMIT sentences. Two of the databases were created such that each sentence in a given concatenation was spoken by the same speaker, whereas the two others mixed speakers within concatenations. In two of the databases, the non-speech percentage of the total signal was 20 %, whereas in two others it was 60 %. We thus had the following four databases:

- DBVAD1 : Mixed speakers, 60% non-speech
- DBVAD2 : Mixed speakers, 20% non-speech
- DBVAD3 : Same speakers, 60% non-speech
- DBVAD4 : Same speakers, 20% non-speech

Further, while the overall amount of non-speech present in the final concatenation was conserved at either 60% or 20%, the apportionment of non-speech before and after each individual sentence varied, being chosen at random from an exponential distribution. This was done in order than the arrival of the sentences resembled a Poisson process, which was thought the most realistic approximation of a real conversation.

We tested each algorithm on each database with an amount of white noise added at levels of SNR between 100 dB and -20 dB.

3.2. Performance measures

Relevant measures of performance were derived from a review of the literature [26] and [27]. The two most important measures of VAD performance are the speech hit rate (HR1) and non-speech hit rate (HR0). These are the fraction of speech frames (resp. non-speech frames) correctly identified as speech frames (resp. non-speech frames):

$$HR0 = \frac{N_{0,0}}{N_0^{ref}} \tag{3.1}$$

$$HR1 = \frac{N_{1,1}}{N_1^{ref}} \tag{3.2}$$

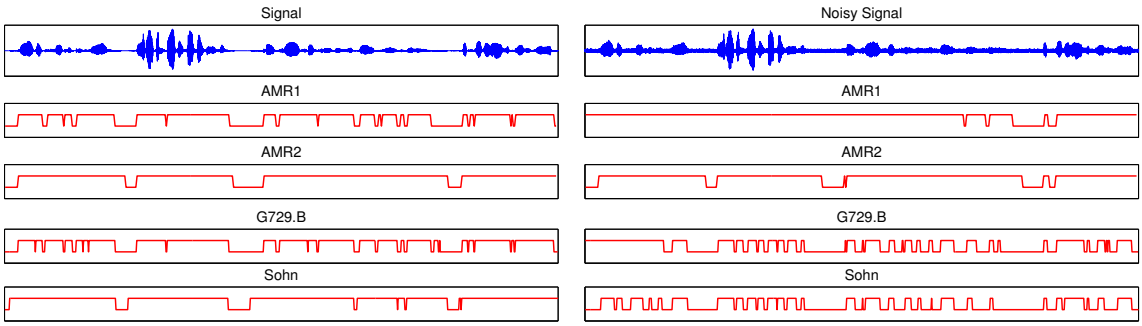
where $N_{0,0}$ is the number of signal frames identified as non-speech by the VAD, N_0^{ref} is the number of signal frames which are actually non-speech according to some reference transcription (in our case provided by our concatenation of TIMIT sentences,) $N_{1,1}$ is the number of signal frames identified as speech, and N_1^{ref} is the number of frames which actually contain speech. Also defined is the the false alarm rate $FAR0 = 1 - HR1$.

Naturally for a good performance in a VAD, one seeks the highest value possible for both of these hit rates.

4. Results

We provide here an example of the VAD decisions produced by the algorithms. We ran each algorithm on a sentence in DBVAD2 (mixed speakers, 20% non-speech time.) In sub-figure 2.6a, we see the output of the algorithms given an input of nearly-clean speech (SNR at 100dB.) Notice that AMR1 and G729.B give strikingly similar speech-nonspeech labeling, the differences being visible only on closer inspection. Further, the output of AMR2 and the algorithm of Sohn are largely similar. AMR2 provides perhaps the closest approximation to the reference speech-nonspeech labeling.

Figure 2.5.: Comparison of labelings provided by VAD algorithms. Four different speakers with 20% non-speech time.



(a) SNR=100dB

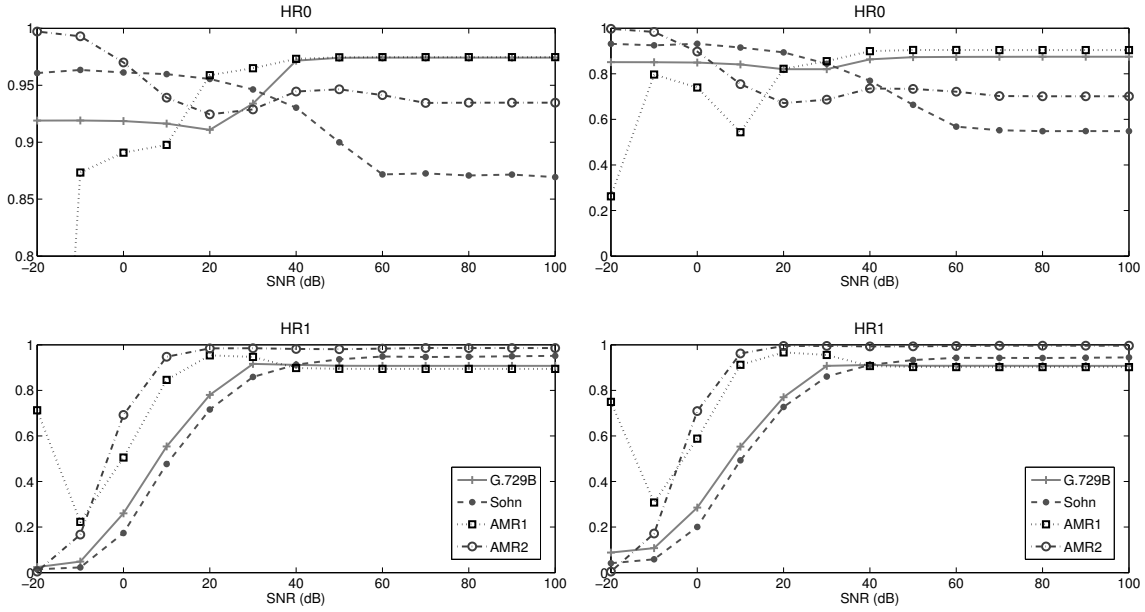
(b) SNR=10dB

The next sub-figure, figure 2.6b shows the results of decreasing the SNR to 10dB. We see that the quality of the labeling has declined, fairly dramatically in some cases. Notice the tendency of the error of the respective algorithms. The AMR1 algorithm tends toward erroneously labeling non-speech as speech, which is probably desirable from a coding perspective, wherein we wish the algorithm to preserve the intelligibility of the signal in the face of uncertainty on the speech-nonspeech decision. In other words, the cost of coding speech with a non-speech compression rate is higher than the cost of coding non-speech with at a speech bit rate. This convention is not respected for the algorithm of Sohn, and strangely, G.729B, which err on the side of classifying all segment as non-speech. Finally, the AMR2 algorithm's output is relatively

unaffected by the increase in SNR, showing its robustness to noise.

One method of representing the noise-robustness of a VAD algorithm is the plotting of non-speech hit rate HR0 and speech hit rate HR1 as a function of SNR. For the following figures we evaluated each algorithm on all four of our TIMIT-derived test databases with SNRs between 100 dB and -20 dB in increments of 10 dB. The first figure, figure 2.7a presents the results testing on DBVAD1. We see that the non-speech hit rate is largely similar for all methods on this database, however the small-scale behavior differs. Whereas HR0 rises with increasing SNR for G.729B and AMR1, it declines for the VAD of Sohn and declines then stabilizes for AMR2. The speech hit rate behaves generally the same for all four methods, rising with increasing SNR as should be expected. Here, the AMR methods show distinct advantages over G.729B, by as much as a 100% increase in improvement (at 10 dB between AMR2 and Sohn.)

Figure 2.6.: VADs tested on DBVAD1 & DBVAD2: Mixed speakers



(a) 60% non-speech time

(b) 20% non-speech time

Looking at figure 2.7b, we see a significant drop in the non-speech hit rates of the Sohn, AMR1, and AMR2 algorithms when using concatenations composed of 20% non-speech by duration rather than 60%. The G.729B algorithm maintains most of its non-speech hit rate in the switch. We see some modest improvement in the speech hit rate of AMR1, on the order of 1 to 10%. The G.729B and Sohn algorithms show some modes improvement at very low SNR, but largely the algorithms maintain the same HR1 performance between the two non-speech time durations.

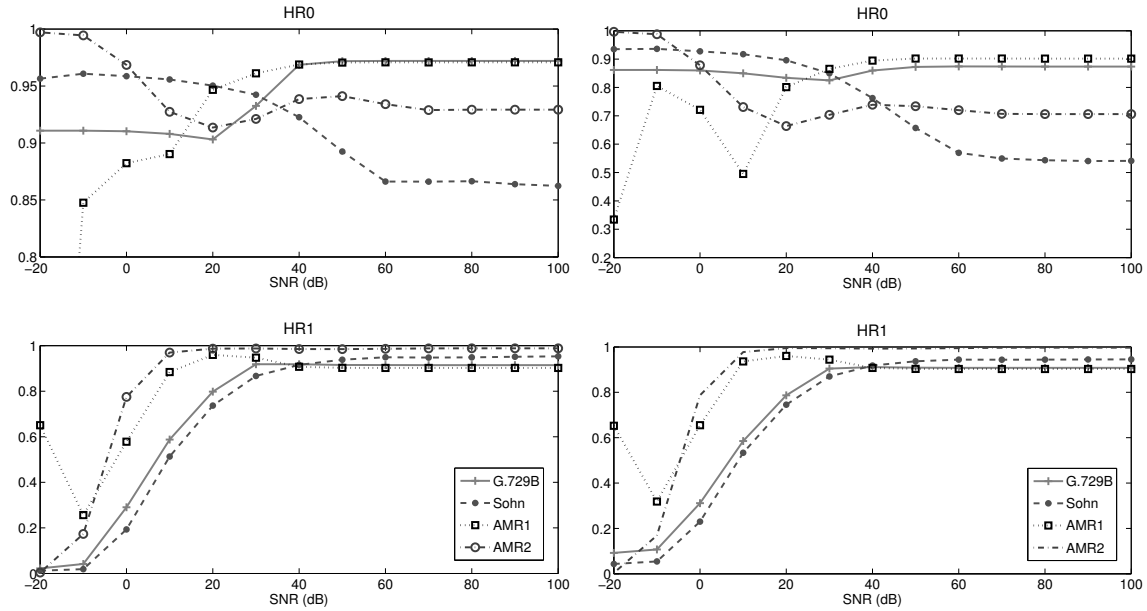
Figures 2.8a and 2.8b depict the results of testing on the DBVAD3 and DBVAD4 databases, wherein each concatenation is composed of four sentences from the same speaker. Comparing these results with those shown in 2.7a and 2.7b, we see that whether the sentences in the concatenations came from mixed or matched speakers, there was essentially no difference in performance.

These hit rates also tie in well with the illustration in figure 2.5. For example, we see the algorithm of Sohn tending to err on the side of non-speech with decreasing SNR. This is reflected in the increasing non-speech hit rate for this algorithm when moving from 100 dB to -20 dB. The non-speech hit rate of AMR1, on the other hand, decreases with decreasing SNR, which is reflected by the very conservative labeling of the speech signal in figure 2.5 as being composed of nearly all speech.

5. Comparison to Other Works

Ying, et. al [25] test these four methods on concatenations of two files from the TIMIT database. They give their results as region-of-operation (ROC) curves, which show the tradeoff between one performance measure (non-speech hit rate) and another (speech hit rate) as a function of a free parameter. However,

Figure 2.7.: VADs tested on DBVAD3 & DBVAD4: Same speakers



(a) 60% non-speech time

(b) 20% non-speech time

G.729B, AMR1, and AMR2 have no free parameters and will therefore appear as points on these graphs. We take the best performance for the algorithm of sohn, which is to say the highest harmonic mean of speech and non-speech hit rate. While exact scores are not given, the following approximations can be made from examining the ROC curves. We compare them with our results on the databases using mixed speakers.

We also compare our results to those found in [27], which tests the algorithms in this paper alongside the author's proprietary VAD. While this author tests on a different database (and thus a direct comparison is impossible,) it is worthwhile to note that the overall behavior of the HR0 and HR1 curves is confirmed in comparing ours with theirs.

6. Conclusion

We implemented and tested four standard VAD algorithms. Our experiments confirmed our intuitions that performance in the form of speech hit rate improves with SNR. A brief discussion demonstrated how a

Table 2.1.: Comparison of our scores to those of [25]

	0 dB		**10 dB**	
Method	HR0	HR1	HR0	HR1
G.729B Ying et. al	80-85	55-60	80-85	80-85
Our G.729B DBVAD1	91	26	92	55
Our G.729B DBVAD2	85	29	84	55
AMR1 Ying et. al	40	85-90	50	85-90
Our AMR1 DBVAD1	89	51	90	85
Our AMR1 DBVAD2	74	59	54	91
AMR2 Ying et. al	45	85-90	38	100
Our AMR2 DBVAD1	97	69	94	95
Our AMR2 DBVAD2	90	71	75	96
Sohn Ying et. al	75-80	75	70	85-90
Our Sohn DBVAD1	96	17	96	48
Our Sohn DBVAD2	93	20	92	50

simple example reflected well the hit rates we obtained. The AMR2 algorithm proved to be the most robust to noise, followed by the AMR1 algorithm. G.729B and Sohn yielded comparable performance, and all algorithms showed similar speech-hit rate behavior with varying SNR.

Chapter 3.

Outcome

My time working at INRIA has been prolific. I have examined the theoretical basis of text-independent phoneme segmentation and voice activity detection, reviewed existing methods and their functioning, including implementing them and evaluating their strengths and weaknesses. Based on a review of speaker segmentation methods, I developed a hypothesis that these methods would work well in phoneme segmentation – especially if combined with simple non-parametric methods already used in phoneme segmentation such as SVF. Based on this I synthesized three algorithms along this line of thought, and evaluated them. The results of this work were significant: I implemented improvements to the algorithms which not only dramatically decreased run times, but also improved on their performance relative to last year. Notably, all algorithms yield performance comparable to that provided by the state of the art in text-independent phone segmentation. Even the simplest among them, DISTBIC-SVF, segments speech well enough to merit publication in the international conference SPECOM 2011 (http://specom.ru/). Taken with the substantial amount of work we did in this area, we were able to succeed in publication of some of our work in a conference paper. Moreover, in light of my implementation and thorough testing of the other phoneme segmentation methods which I developed, this report will serve as the basis for an upcoming publication in an peer-reviewed international journal. Besides segmentation, my work in VAD has provided the team with a framework for evaluating different VAD algorithms, in addition to my implementation of four VAD standards, both of which will be useful for future research.

I have developed my technical skills, including improvement of my ability to program in LaTeX and to work in a command-line environment such as Linux. I also significantly strengthened my knowledge of Matlab, and achieved greater familiarity with C programming and shell scripting, which will be important assets in my career working in speech processing. I also learned how to work with databases of speech and design and carry out evaluations of speech processing algorithms, becoming more knowledgeable of experimental design and interpretation of results in this field.

The work I have done in speech processing at INRIA gave me experience and expanded my abilities in this domain, and added value to my profile as an engineer. Furthermore, I have experienced working in a team over a long duration for a common objective in the context of a research organization. Not only this, it was also thanks to my experience gained working on these projects that I I succeeded in obtaining a PhD position in speech recognition at the Karlsruhe Institute of Technology under the supervision of Alexander Waibel, whose groups (in Karlsruhe and at Carnegie Mellon) are world-renowned in speech processing. Thus my efforts at INRIA have helped secure the next critical step in achieving my vision of a career in speech processing research.

Appendix

Appendix A.

Explanation of Methods and Development in Work

1. Tools

In the my work on phoneme segmentation and VAD the primary software used was Matlab within a Linux environment. This has opened opportunities for shell scripting to automate some of the more routine tasks. A small amount of work has been done in C due to the fact that the source code for some of the standard methods I sought to implement is in this language. While C, a compiled language, undoubtedly boasts faster execution times than Matlab, an interpreted language, the large majority of development last year was in Matlab and so this environment was used to more easily continue this development and testing.

One toolbox for Matlab which proved very useful in this work was Voicebox. This set of Matlab functions, developed by Imperial College, is oriented toward speech processing applications. Included inside were several notable and essential functions:

- A function for computing the Mel-Frequency Cepstral Coefficients (MFCCs), a standard feature used in the segmentation algorithms
- A function for generating a random subset of numbers within a given range, useful for the generation of test databases from TIMIT
- A function for framing signals and producing the time indices of the frame centers
- A function for generating various types of window functions
- Functions for reading and writing .wav files
- Functions for converting between the *mel* and *frequency* scales
- A function for generating all possible permutation of numbers in a given range

Also very useful was the fact that Voicebox included a function to compute the VAD of Sohn, a VAD standard we wished to implement. Several other functions from the toolbox played more minor roles.

2. Applying Greater Rigor in Phoneme Segmentation

While reviewing the phoneme segmentation in preparation for running experiments on them, I saw the need for sevearl changes in to the algorithms to bring them into line with standard practices in speech processing.

One area which needed changing was in the manner in which the ΔBIC test was implemented in those algorithms using it as a second stage. As one instance of this, we take DISTBIC-SVF, a two-pass algorithm. In this method we have a first stage wherein heuristics generate candidate transition points. Due to the use of framing in the analysis for this stage, these candidates occur only at discrete locations at a lower resolution than the original signal. Afterwards, a second stage re-extracts framed features and uses the candidates as boundaries of dynamic windows. In last year's work, we converted the lower-resolution candidate indices to time indices in the scale of the original signal, then for each window in the dynamic windowing process we re-extracted features. Not only was this slow – as the extraction of features is, relatively speaking, one of the most time-consuming processes executed in the algorithm – it also lead to paradoxes. One such paradox was the fact that a dynamic window could contain more feature vectors than its two subwindows combined. In order to correct this, I modified the algorithm to convert the indices of the first feature scale to the scale of the spacing of the second pass of feature extraction. This involved choices pertaining to what we consider a candidate index to refer to – the beginning, center, or end of the feature window for the first pass – and how to align it to the feature windows of the second pass. For instance, consider the case where the features for the first pass had a frame step of 10 points and a frame length of 24 points. Suppose the first pass returns candidates at $\{1, 2, 5, 12, 14, 25, 31\}$. Now we have the following results from the three alignments:

$$\begin{aligned} \text{Left align:} &\quad \{0, 10, 40, 110, 130, 240, 300\} \\ \text{Center align:} &\quad \{12, 22, 52, 122, 142, 252, 312\} \\ \text{Right align:} &\quad \{10, 20, 50, 120, 140, 250, 310\} \end{aligned}$$

Another factor which was discovered was the different way that the Mel Frequency Cepstral Coefficients (MFCCs) and Human Factor Cepstral Coefficients (HFCCs) were implemented. MFCCs and HFCCs are two conceptually similar methods for modeling sound. In each, we first take the Fourier transform of window of the signal to obtain the power spectrum, the map the power spectrum to the mel-scale using triangular overlapping filters. It is in this latter step that MFCCs and HFCCs differ, as the bandwidth of the MFCC filters is based on the filter spacing, whereas the bandwidth of HFCC filters is decoupled and based on the theory of sound processing of the human ear. Next the log is taken of the powers at each mel frequency, followed by the discrete cosine transform of these powers.
Whereas the algorithm which generated the MFCCs was set up with 29 filters in its filter bank and used a number of points in its Fourier transforms (FFT) equal to the size of the window, I had implemented the HFCCs with 32 filters in the filter bank and a number of FFT points fixed at 512. To harmonize these two methods, I settled on a compromise of 29 filters since this is also the number of filters cited in the founding paper of the HFCCs, along with a number of FFT points equal to the number of points in the feature window rounded to the next exponent of 2: $NFFT = 2^{ceil(\log_2 L)}$ where $ceil(\cdot)$ is the ceiling function and L is the length of the feature window under consideration.

Last, I made the decision to use diagonal covariance matrices in the ΔBIC statistic for all methods using ΔBIC. This choice was motivated by the often small size of the BIC analysis windows. Since we set out algorithm to generate 12 features in each feature vector, a given analysis window is $N \times 12$. Often times, N is less than 20, which gave rise to concerns that we may be estimating statistics with insufficient data. Changing the number of figures to estimate should also change the penalty factor in the ΔBIC statistic. Whereas with full covariance matrices this penalty was $\frac{1}{2}(p + \frac{1}{2}p(p+1))\log(N_Z)$ with $p = 12$ being the dimension of the feature vectors, the new penalty is simply $2p\log(N_Z)$.

These changes necessitated the running of new experiments for all methods devised last year which we are to include in the paper.

3. Evaluation Paradigm for Phoneme Segmentation and Associated Work

3.1. TIMIT Database

I wished to evaluate the quality of the results produced by our algorithms in a way that is readily comparable to that of other algorithms. The evaluation metric of choice is usually a comparison to a manually annotated speech corpus. For this TIMIT was used, a corpus of read speech that is the most widely used for phoneme segmentation experiments [10]. Comprised of 6300 sentences spoken by 192 female and 438 male speakers drawn from 8 different American English dialects, the corpus is divided into a 'train' and a 'test' set. For each sentence in the database there are four files: 1) a .wav file containing the speech signal itself, 2) a .phn file containing a sequential list of the beginning and end times of each phoneme as well as the ARPAbet phoneme symbol itself, 3) a .wrd file specifying the beginning and end times of each word in the sentence, and 4) a .txt file being the sentence text.

3.2. Evaluating a Phoneme Segmentation Algorithm

I wished to evaluate the quality of the results produced by our algorithms in a way that is readily comparable to that of other algorithms. The evaluation metric of choice is usually a comparison to a manually annotated speech corpus. For this TIMIT was used, a corpus of read speech that is the most widely used for phoneme segmentation experiments [10]. Comprised of 6300 sentences spoken by 192 female and 438 male speakers drawn from 8 different American English dialects, the corpus is divided into a 'train' and a 'test' set. For each sentence in the database there are four files: 1) a .wav file containing the speech signal itself, 2) a .phn file containing a sequential list of the beginning and end times of each phoneme as well as the ARPAbet phoneme symbol itself, 3) a .wrd file specifying the beginning and end times of each word in the sentence, and 4) a .txt file being the sentence text.

The method I developed last year for evaluating a given phoneme segmentation algorithm on the TIMIT database was relatively straightforward. The first step was the creation of a Matlab function for each segmentation algorithm. This function accepted as inputs the various free parameters controlling it, as well as the path to the speech (.wav) file and the path to the associated phoneme index (.phn) file. The script for evaluating this algorithm contained at its head a set of arrays containing the values of the free parameters that would be used to test the function. A cell array is loaded which contains the paths to all .wav and .phn

files in the desired database. This was followed by a set of nested loops with the function at its core. Each loop stepped through a different value in the array. An example might be as follows:

Algorithm 1 Example evaluation algorithm

These are the arrays containing possible values for the parameters
$[WAVS, PHNS] = \text{load}('TIMITtrain.mat')$;
$alphaset = [0, 0.1, 0.2, 0.5, 0.75, 1]$;
$fstepset = [4, 16, 32, 48, 64]$;
$flenset = [48, 64, 128, 256]$;
for $i = 1$ to length$(alphaset)$ do
  for $j = 1$ to length$(fstepset)$ do
    for $k = 1$ to length$(flenset)$ do
      for $g = 1$ to length$(WAVS)$ do
        $wavpath = WAVS\{g\}$;
        $phnpath = PHNS\{g\}$;
        $performances\{i, j, l\} = \text{functionDACDec}(wavpath, phnpath, ...$
          $alphaset(i), fstepset(j), flenset(k))$
      end
    end
  end
end

This method of evaluating performances left the output performances in an awkward structure. Futher, it was not possible to run the function for just some of the possible values. Last, this sequential method of evaluation utilized but a fraction of a mult-core computer's processing capability.

4. Parallelization

The experiments involved in development and testing of the methods examined last year and this year principally rely on testing a given method on several hundred or even several thousand files from the TIMIT corpus. This can be very time consuming. To speed up these experiments, I adapted my code to make use of a parallelization toolset for Matlab enabling tasks to be executed simultaneously on multiple cores.

This year's experiments were greatly aided by my adapting my code to make use of parallelization, enabling tasks to be executed simultaneously on multiple cores thus exploiting a much greater amount of a workstation's computing. I used the Multicore file package available on the Mathworks file exchange, and adapted my code to fit within this paradigm. With this, one creates a cell array of parameters, specifies the function, and hands off processing to a multicore program which assigns work to each core of the computer.

A new structure of the program looked something like this:

Using this technique on my workstation, possessing 8 cores, reduced run times by as much as 88%. This change has greatly reduced the processing time required for some experiments, and thus enabled testing of more hypotheses and exploration of more ideas related to the behavior of the algorithms under development.

5. Evaluation Paradigm for VAD and Associated Work

5.1. Adaptation of TIMIT for the Creation of a VAD Evaluation Database

The first task in implementing standard VAD methods and potentially creating new ones is the setup of a system for running and evaluating these algorithms. I was tasked with adapting the TIMIT database, which is traditionally used for phoneme segmentation experiments, to act as a reference database for VAD. The TIMIT database consists of several thousand sentences spoken in a studio environment with some very brief silence before and after the speech. In order to turn these types of files into something which tests an algorithm's ability to distinguish speech from silence and test its robustness to noise, I created programs which concatenate signals from individual TIMIT files, adding in a random amount of silence of a length drawn from an exponential distribution. In this way the spoken phrases resemble random events in a Poisson process with the corresponding inter-arrival time. The Matlab scripts I created accept as input several parameters which control the type of database created. One may specify the number of "concatenations" desired, as well as the number of sentences in each concatenation, as well as whether the sentences in each

Algorithm 2 Example parallelized evaluation algorithm

These are the arrays containing possible values for the parameters
$[WAVS, PHNS] = \text{load}('TIMITtrain.mat')$;
$alphaset = [0, 0.1, 0.2, 0.5, 0.75, 1]$;
$fstepset = [4, 16, 32, 48, 64]$;
$flenset = [48, 64, 128, 256]$;
$k = 0$;
for $i = 1$ to length$(alphaset)$ do
  for $j = 1$ to length$(fstepset)$ do
    for $k = 1$ to length$(flenset)$ do
      for $g = 1$ to length$(WAVS)$ do
        $k = k + 1$;
        $wavpath = WAVS\{g\}$;
        $phnpath = PHNS\{g\}$;
        $parameterCell\{1, k\} = \{wavpath, phnpath, alphaset(i), fstepset(j), flenset(k)\}$
      end
    end
  end
end
$resultCell = \text{startmulticoremaster}(@functionDACDec, parameterCell)$;

concatenation should be drawn from the same speaker. In addition, for each concatenation a text file of extension '.bnd' (for our purposes a boundary file,) is created specifying the start and end times of the segments containing speech. This is created based on the text file accompanying each sentence which specifies the start and end times of each phoneme. Additionally one may select whether the epenthetic silence sometimes found in TIMIT sentences is counted as silence in the concatenation. In this way I have created tools for creating richly customizable VAD databases from TIMIT.

5.2. Implementation of VAD Scoring

Next, I created algorithms to score the performance of the various VAD methods to be implemented. We saw earlier the definitions of the performance measures used in these experiments, which were taken from the most common measures found in the literature.

It was necessary to develop a program which would accept a variety of specifications for the speech and non-speech labels when going about scoring the performance of these algorithms. For instance, the text files associated to the TIMIT-based concatenations specify merely the start and stop times of the speech segments. On the other hand, most of the VADs considered produce a frame-by-frame classification. A conversion between the two types was thus necessary to evaluate the labeling produced by the VAD.

5.3. Evaluationg a given VAD

The evaluation paradigm for VAD largely resembled the one for evaluation of segmentation algorithms. I used the same parallelization which sped up the task of evaluating segmentation algorithms. As stated before, we evaluate a given VAD method under different conditions. One of these conditions is Signal to Noise Ratio (SNR.) The test files created for the VAD evaluation database are essentially noise-free, being composed of concatenations of TIMIT speech files which consist of studio-quality recordings of speech. Therefore noise is added during the testing process. For these experiments, the noise was limited to white gaussian noise. Due to the random nature of the white noise, I thought it prudent to perform multiple evaluations of the same VAD on the same files, and to take an average performance afterwards.

The evaluation proceeds as follows. First, the evaluation script defines the values of SNR values to test, along with specifying the number of evaluations and the number of files over which to test (which cannot in any case exceed the number of files in the database.) Next, a loop creates a parameter cell containing the path to the test concatenation, its associated boundary file, the signal to noise ratio, and various indices useful in debugging. Then the multi-core algorithm evaluates a function using theses parameters.

The aforementioned function is formatted similarly for all VADs. It accepts as input the path to the concatenation under test and the file specifying its boundaries, as well as SNR. Inside the function the concatenation .wav file is loaded into memory along with its boundaries. The power of the speech segments is then calculated, and the requisite noise power is computed based on this. White noise is then generated at this power and added to the signal. Then, depending on the type of algorithm, this noisy signal may inputted directly to the VAD (ex.: VAD of Sohn) or saved as a temporary file, the path to which will be taken by the VAD as input (ex.: G.729B.)

Appendix B.

Using ML Estimators in Maximum Log-Likelihood Ratios for ΔBIC

The basis of the methods for phoneme segmentation we have developed is the use of the ΔBIC test to judge whether a transition has occurred or not. In judging between two hypotheses, the ΔBIC test is based on a penalized log likelihood ratio. The following is a theoretical derivation of the basis for the form of the expression used in the literature.

1. Description of Problem

We begin with a set of data $X = \{x_i\}_{i=0}^{N}$ where x_i are k-dimensional feature data. Thus X is an $N \times k$ matrix with x_i a $1 \times k$ feature vector. We partition the data at feature vector index q. Thus we have $X = \{x_i\}_{i=0}^{N_X}$ being the full segment, $X_1 = \{x_i\}_{i=0}^{N_{X_1}}$ being one subsegment, and $X_2 = \{x_i\}_{i=N_{X_1}+1}^{N_X}$ being the other subsegment. We form two hypotheses:

- H_0: The data adheres to model M_0 meaning that $X \sim N(\mu_X, \Sigma_X)$ meaning that the two segments contain features drawn from the same distribution
- H_1: The data adheres to model M_1, in that $X_1 \sim N(\mu_{X_1}, \Sigma_{X_1})$ and $X_2 \sim N(\mu_{X_2}, \Sigma_{X_2})$ meaning that the segments are drawn from separate distributions and thus that there is a transition

Comparing two models M_0 and M_1 in the context of Bayesian model selection implies the choice of the model with the higher posterior probability. We have the posterior likelihood ratio:

$$\frac{p(M_0|x)}{p(M_1|x)} = \frac{p(x|M_0)p(M_0)}{p(x|M_1)P(M_1)} = BF \times \text{prior likelihood}$$
$$BF = \frac{\int_{\Theta_0} p(x|\theta_0, M_0)p(\theta_0|M_0)d\theta_0}{\int_{\Theta_1} p(x|\theta_1, M_1)p(\theta_1|M_1)d\theta_1} \tag{1.1}$$

BF is the ratio of marginal likelihoods between the two models. $\theta_0 \in \Theta_0$ and $\theta_1 \in \Theta_1$ are the parameters of each model. In practice it is difficult to compute the marginal likelihoods given in equation 1.1. This might be done for example with Gibbs sampling. [28] states that if we consider that the observations x to be drawn from an exponential distribution, the BIC, or Bayesian Information Criterion "results as an easily calculated and asymptotically optimal method for estimating the best model using only MLE of the parameter vectors θ_0 and θ_1." This is a long-sample approximation to the BF assuming flat priors. The BIC is given as

$$BIC = -2 \log l(\hat{\theta}) + K \log(n) \tag{1.2}$$

where $l(\hat{\theta})$ is the likelihood function evaluated with the MLE of the parameters, n is the sample size, and K is the number of free parameters estimated. The likelihood gives an indication of the quality of fit of the model to the data, while the second term penalizes the model complexity.

ΔBIC, the difference between the BIC value of the two models, determines whether the model M_1, with segment X_1 following the multivariate Gaussian mixture $N(\theta_1)$ and $X_2 \sim N(\theta_2)$ or model M_0 with all data X following the univariate gaussian $N(\theta_0)$ best fits the data. The ΔBIC is given by:

$$\Delta BIC(M_1, M_0) = BIC(M_1) - BIC(M_0) \tag{1.3}$$

If $\Delta BIC(M_1, M_0) < 0$, we conclude that the data is better described by two gaussians and thus that a transition is present.

We next derive the ΔBIC in the case of unidimensional data as a preparation for the multivariate case.

2. Case of univariate data

We first consider the case of the partitioned data (the hypothesis that a transition exists,) which is represented by $BIC(M_1)$:

$$\begin{aligned} BIC(M_1) &= -2\log l(X|M_1) + K\log(n) \\ &= -2L(X|M_1) + P_1 \end{aligned} \tag{2.1}$$

where we define the log-likelihood $L(\cdot) = \log(l(\cdot))$.

The penalty term is

$$K\log(n) \tag{2.2}$$

with n being the length of the data. Here we have $n = N_X$ data and four parameters – μ_{X_1}, μ_{X_2}, Σ_{X_1}, and Σ_{X_2} – so we have $P_1 = 4\log(N_X)$.

$$\begin{aligned} L(X|M_1) &= \log(l(X|M_1)) = \log(l(X_1|\theta_{X_1})l(X_2|\theta_{X_2})) \\ &= L(X_1|\theta_{X_1}) + L(X_2|\theta_{X_2}) \end{aligned} \tag{2.3}$$

We assume the data are independent and identically distributed (iid) according to a univariate normal distribution:

$$\begin{aligned} l(X_1|\theta_{X_1}) &= \prod_{i=1}^{N_{X_1}} l(x_i|\theta_{X_1}) \\ &= \prod_{i=1}^{N_{X_1}} (2\pi)^{-1/2}(\sigma_{X_1}^2)^{-1/2}\exp\{-\frac{(x_i-\mu_{X_1})^2}{2\sigma_{X_1}^2}\} \\ &= (2\pi)^{-N_{X_1}/2}(\sigma_{X_1}^2)^{-N_{X_1}/2}\exp\{-\frac{\sum_{i=1}^{N_{X_1}}(x_i-\mu_{X_1})^2}{2\sigma_{X_1}^2}\} \end{aligned} \tag{2.4}$$

and for the log-likelihood:

$$L(X_1|\theta_{X_1}) = -\frac{N_{X_1}}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log(\sigma_{X_1}^2) - \frac{\sum_{i=1}^{N_{X_1}}(x_i-\mu_{X_1})^2}{2\sigma_{X_1}^2} \tag{2.5}$$

We also have that $\sum_{z=1}^{N}(z_i-\mu_Z)^2 = N(\bar{Z^2} - 2\bar{Z}\mu_Z + \mu_Z^2)$ where $\bar{Z}$ is the mean of all points z_i from $i = 1 \dots N$ and $\bar{Z^2}$ is the mean of their squares. We can use this substitution in equation 2.5:

$$L(X_1|\theta_{X_1}) = -\frac{N_{X_1}}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log(\sigma_{X_1}^2) - \frac{N_{X_1}(\bar{X_1^2} - 2\bar{X_1}\mu_{X_1} + {\mu_{X_1}}^2)}{2(\sigma_{X_1}^2)} \tag{2.6}$$

We now substitute the maximum likelihood estimators for the mean and variance, which are $\hat{\mu} = \bar{x}_1$, the sample mean, and $\hat{\sigma}^2 = \bar{x_1^2} - \bar{x}_1^2$. This simplifies the expression.

$$\begin{aligned} L(X_1|\hat{\theta}_{X_1}) &= -\frac{N_{X_1}}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log(\hat{\sigma}_{X_1}^2) - N_{X_1}\frac{\bar{X_1^2} - 2\bar{X_1^2} + \bar{X_1^2}}{2(\bar{X_1^2}) - \bar{X_1^2}} \\ &= -\frac{N_{X_1}}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log(\hat{\sigma}_{X_1}^2) - N_{X_1}\frac{\bar{x_1^2} - \bar{X_1^2}}{2(\bar{X_1^2} - \bar{X_1^2})} \\ &= -\frac{N_{X_1}}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log(\hat{\sigma}_{X_1}^2) - \frac{N_{X_1}}{2} \end{aligned} \tag{2.7}$$

The same holds true for $L(X_2|\theta_{X_1})$, where we proceed from the initial expression for the likelihood, noting the change in the subscripts and the fact that $N_X - N_{X_1} = N_{X_2}$:

$$l(X_2|\theta_{X_2}) = \prod_{i=N_{X_1}}^{N_X} l(x_i|\theta_{X_2}) \tag{2.8}$$

With the steps coming after being the same as for $L(X_1|\theta_{X_1})$ we are lead to the following expression for the log-likelihood of X_2 under the ML estimates:

$$L(X_2|\hat{\theta}_{X_2}) = -\frac{N_{X_2}}{2}\log(2\pi) - \frac{N_{X_2}}{2}\log(\hat{\sigma}^2_{X_2}) - \frac{N_{X_2}}{2} \tag{2.9}$$

Thus we then have the following for $L(X|M_1)$:

$$\begin{aligned} L(X|M_1) &= -\frac{N_{X_1}}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log(\hat{\sigma}^2_{X_1}) - \frac{N_{X_1}}{2} - \frac{N_{X_2}}{2}\log(2\pi) \\ &\quad - \frac{N_{X_2}}{2}\log(\hat{\sigma}^2_{X_2}) - \frac{N_{X_2}}{2} \\ &= -\frac{N_X}{2}\log(2\pi) - \frac{N_X}{2} - \frac{N_{X_1}}{2}\log(\hat{\sigma}^2_{X_1}) - \frac{N_{X_2}}{2}\log(\hat{\sigma}^2_{X_2}) \end{aligned} \tag{2.10}$$

Now for the non-partitioned data, the BIC statistic is
$BIC(M_0) = -2L(X|M_0) + P_0$:

$$L(X|M_0) = L(X|\theta_X) = -\frac{N_X}{2}\log(2\pi) - \frac{N_X}{2}\log(\hat{\sigma}^2_X) - \frac{N_X}{2} \tag{2.11}$$

For the penalty P_0, we have only two parameters to estimate, μ_X and σ_X, so the penalty is $P_0 = 2\log(N_X)$.

Thus all three likelihood terms in the following expression follow the above structure:

$$\begin{aligned} \Delta BIC(M_1, M_0) &= BIC(M_1) - BIC(M_0) \\ &= -2L(X|M_1) + P_1 + 2L(X|M_0) - P_0 \\ &= -2\{-\frac{N_X}{2}\log(2\pi) - \frac{N_X}{2} - \frac{N_{X_1}}{2}\log(\hat{\sigma}^2_{X_1}) - \frac{N_{X_2}}{2}\log(\hat{\sigma}^2_{X_2})\} \\ &\quad + 2\{-\frac{N_X}{2}\log(2\pi) - \frac{N_X}{2} - \frac{N_X}{2}\log(\hat{\sigma}^2_X)\} \\ &\quad + 4\log(N_X) - 2\log(N_X) \\ &= N_X\log(2\pi) + N_X + N_{X_1}\log(\hat{\sigma}^2_{X_1}) + N_{X_2}\log(\hat{\sigma}^2_{X_2}) \\ &\quad - N_X\log(2\pi) - N_X\log(\hat{\sigma}^2_X) - N_X + 2\log(N_X) \\ &= N_{X_1}\log(\hat{\sigma}^2_{X_1}) + N_{X_2}\log(\hat{\sigma}^2_{X_2}) - N_X\log(\hat{\sigma}^2_X) + 2\log(N_X) \end{aligned} \tag{2.12}$$

3. Case of Multivariate Data

In this case, we are deciding between the hypothesis that $X_1 \sim N(\mu_{X_1}, \Sigma_{X_1})$ and $X_2 \sim N(\mu_{X_2}, \Sigma_{X_2})$, and the hypothesis that $X \sim N(\mu_X, \Sigma_X)$ where μ is a mean vector and Σ is a full covariance matrix. Equations 1.3, 2.1, and 2.2 give us:

$$\Delta BIC = -2L(X|M1) + 2L(X|M0) + K_{M_1}\log(N_X) - K_{M_0}\log(N_X) \tag{3.1}$$

Because we are dealing now with mean vectors and covariance matrices, the number of parameters to be estimated is increased. For a single gaussian, the number of parameters to be estimated is $K_g = k + \frac{1}{2}k(k+1)$, since we have $1 \times k$-dimensional mean vectors plus $k \times k$-dimensional covariance matrices. In the case of model M_1 with two gaussians, $K_{M_1} = 2K_g$, whereas M_0, the assumption of one gaussian, has $K_{M_0} = K_g$. Adding the two penalties we have:

$$\Delta BIC = -2L(X|M_1) + 2L(X|M_0) + (k + \frac{1}{2}k(k+1))\log(N_X) \tag{3.2}$$

Equation 2.3 remains valid, and we will proceed from it in the same way as in the univariate case in order to derive the expression for ΔBIC. Equation 2.4 is different due to the multivariate pdf:

$$
\begin{aligned}
l(X_1|\theta_{X_1}) &= \prod_{i=1}^{N_{X_1}} l(x_i|\theta_{X_1}) \\
&= \prod_{i=1}^{N_{X_1}} (2\pi)^{-k/2}(|\Sigma_{X_1}|)^{-1/2} \exp\{-\frac{1}{2}(x_i-\mu_{X_1})^\top \Sigma_{X_1}^{-1}(x-\mu_{X_1}\} \\
&= (2\pi)^{-kN_{X_1}/2}(\Sigma_{X_1})^{-N_{X_1}/2} \exp\{-\frac{1}{2}\sum_{i=1}^{N_{X_1}}(x_i-\mu_{X_1})^\top \Sigma_{X_1}^{-1}(x-\mu_{X_1}\}
\end{aligned}
\tag{3.3}
$$

Then the log-likelihood is:

$$
\begin{aligned}
L(X_1|\theta_{X_1}) = &-\frac{dN_x}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log|\Sigma_{X_1}| \\
&- \frac{1}{2}\sum_{i=1}^{N_{X_1}}(x_i-\mu_{X_1})^\top \Sigma_{X_1}^{-1}(x-\mu_{X_1}\}
\end{aligned}
\tag{3.4}
$$

We know that if $\Sigma_{X_1} = \hat{\Sigma}_{X_1}$ that is, equal to its ML estimator the sample dispersion matrix, we have the following simplification from [29]:

$$
\begin{aligned}
&\sum_{i=1}^{N_{X_1}}(x_i-\mu_{X_1})^\top \Sigma_{X_1}^{-1}(x_i-\mu_{X_1}) \\
&= \text{tr}\{\Sigma_{X_1}^{-1}\sum_{i=1}^{N_{X_1}}(x_i-\mu_{X_1})(x_i-\mu_{X_1})^\top\} \\
&= kN_{X_1}
\end{aligned}
\tag{3.5}
$$

Therefore equation 3.4 simplifies to:

$$
L(X_1|\theta_{X_1}) = -\frac{dN_x}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log|\Sigma_{X_1}| - \frac{kN_{X_1}}{2} \tag{3.6}
$$

With a similar simplification taking place for $L(X_2|\theta_{X_2})$ and $L(X|\theta_X)$, ΔBIC reduces to:

$$
\begin{aligned}
\Delta BIC = &-2\{-\frac{dN_{X_1}}{2}\log(2\pi) - \frac{N_{X_1}}{2}\log|\Sigma_{X_1}| - \frac{kN_{X_1}}{2}\} \\
&- 2\{-\frac{dN_{X_2}}{2}\log(2\pi) - \frac{N_{X_2}}{2}\log|\Sigma_{X_2}| - \frac{kN_{X_1}}{2}\} \\
&+ 2\{-\frac{dN_X}{2}\log(2\pi) - \frac{N_X}{2}\log|\Sigma_X| - \frac{kN_X}{2}\} + (k+\frac{1}{2}k(k+1))\log(N_X)
\end{aligned}
\tag{3.7}
$$

Since $N_X = N_{X_1} + N_{X_2}$, the first and third terms within each of the three braces cancel, as well as the factors and divisors of two, leaving the following final simplification:

$$
\begin{aligned}
\Delta BIC = &N_{X_1}\log|\Sigma_{X_1}| + N_{X_2}\log|\Sigma_{X_2}| - N_X\log|\Sigma_X| \\
&+ (k+\frac{1}{2}k(k+1))\log(N_X)
\end{aligned}
\tag{3.8}
$$

with $\Delta BIC < 0$ indicating a transition.

4. Comparison to expression found in Delacourt, 2000

In [5], Delacourt gives the following form for the ΔBIC :

$$
\Delta BIC = -R + \lambda P < 0 \qquad \text{indicates transition} \tag{4.1}
$$

where R and P are given:

$$R = \frac{N_X}{2}\log|\Sigma_X| - \frac{N_{X_1}}{2}\log|\Sigma_{X_1}| - \frac{N_{X_2}}{2}\log|\Sigma_{X_2}|$$
$$P = \frac{1}{2}(k + \frac{1}{2}k(k+1))\log(N_X)$$

Simply multiplying both R and P results in the same expression as found in 3.8

5. Case of diagonal covariance matrices

If we restrict the distributions in our statistical test to use only diagonal covariance matrices, we will have a different number of parameters to estimate. In effect, we will be estimating the multivariate variance of the data, which is a vector of the same dimension k as the data. Because there are two gaussians in M_1, the penalty for $BIC(M_1)$ is $2k\log(X)$. For M_0 we have $k\log(X)$. For $\Delta BIC(M_1, M_0)$ the penalty of M_0 is subtracted from that of M_1, yielding the combined penalty of

$$P = k\log(X) \tag{5.1}$$

Appendix C.

Recursive mean and variance for OCD-Chen

One-Changepoint-Detection via the method of Chen [15] (OCD-Chen) is a method used in Divide-and-Conquer (DACDec) for detecting single change points as a method for phoneme segmentation. The method relies on computing the ΔBIC statistic for progressive configurations of windows and subwindows. If each calculation of the statistic is made independently, this repetitive calculation is time-consuming. On the other hand, an iterative approach may save many computational cycles. The following is a derivation of this iterative approach and an examination of the computational cost-savings.

1. OCD-Chen

As described in section 5.1, OCD-Chen seeks a detection of a single changepoint in analysis window Z of length N_Z. We divide Z into subwindows X_i and Y_i with i being the feature frame (henceforth referred to simply as a data vector or data point) dividing the two subwindows. We start out with $i = d + 1$, where d is the dimension of each data vector (normally the number of cepstral coefficients). i is then incremented up to $N_Z - d - 2$. At each step we compute the covariance matrices for X_i and Y_i to form part of the ΔBIC statistic (the covariance matrix of Z remaining the same for all iterations.) From the collection of $\{\Delta BIC(i)\}_{i=d+1}^{N-d-2}$ we choose i which gives maximum $\Delta BIC(i)$ as the change point. We summarize with the above as "standard" OCD-Chen in the algorithm 3.

Algorithm 3 Standard OCD-Chen

(1) $N_Z = \text{length}(Z)$;
(2) $\sigma_Z^2 = \text{var}(Z)$;
(3) Take the determinant of the covariance in the diagonal case
(4) the product of the elements in the variance vector:
(5) $D_0 = \prod_{j=1}^{d} \sigma_{Z_i}^2(j)$;
(6) for $i = d + 1$ to $N_Z - d - 2$ do
(7) $\quad X_i = Z(1 : i, :)$; $\quad X_i$ is an $i \times d$ matrix
(8) $\quad Y_i = Z(i + 1 : N_Z, :)$; $\quad Y_i$ is an $(N_Z - i) \times d$ matrix
(10) $\quad \sigma_{X_i}^2 = \text{var}(X_i)$; $\quad \sigma_{X_i}^2$ is an $1 \times d$ vector
(11) $\quad \sigma_{Y_i}^2 = \text{var}(Y_i)$; $\quad \sigma_{Y_i}^2$ is an $1 \times d$ vector
(12) $\quad D_1 = \prod_{j=1}^{d} \sigma_{X_i}^2(j)$
(13) $\quad D_2 = \prod_{j=1}^{d} \sigma_{Y_i}^2(j)$; ;
(14) $\quad \Delta BIC(i) = N_Z \log D_0 - i \log D_1 - (N_Z - i) \log D_2 - \lambda P$;
(15) end
(16) $i_{change} = \arg\max_i \Delta BIC(i)$

1.1. Computational Cost Analysis

We neglect the costs associated with computing the length of Z. The costs of the products on lines 5, 12, and 13 of Algorithm 3 are invariant, being equal to $K_p = d \times c_{mult}$ with c_{mult} being the cost of a multiplication. Computing the ΔBIC is also invariant, equal to K_{BIC}.

Computing the variance implies computing the mean. The cost of computing the mean is as follows. The mean of Z consists of $N_Z \times d$ additions of cost c_{add} followed by a divsion by N_Z of cost c_{div}. Therefore for a mean of Z we have a cost of

$$K_{mean}^{Z} = d \times (N_Z \times c_{add} + c_{div}) \tag{1.1}$$

The means of X_i and Y_i will depend on their lengths. Now variance is computed as the sum of the squares of the differences of the data from the mean. Again being reminded of vectorization, we assume each subtraction having cost $c_{sub} \equiv c_{add}$, the square being essentially a multiplication of cost c_{mult}, the additions in

the summation being of cost c_{add} and the division by the number of elements being of cost c_div. Therefore the cost of computing the variance is:

$$\begin{aligned} K_{var}^{Z} &= N_Z \times d \times c_{add} + N_Z \times d \times c_{mult} + N_Z \times d \times c_{add} + d \times c_{div} + K_{mean}^{Z} \\ &= d \times (N_Z \times (3c_{add} + c_{mult}) + 2c_{div}) \end{aligned} \tag{1.2}$$

Since at any point in the iteration of our algorithm the length of X_i plus the length of Y_i equals the length of Z, $K_{var}^{X_i} + K_{var}^{Y_i} = K_{var}^{Z}$. Therefore for our standard OCD-Chen algorithm we have the following cost:

$$K_{OCD-Chen} = K_{var}^{Z} + K_p + (N_Z - 2d - 2) \times (2K_p + K_{var}^{Z} + K_{BIC}) \tag{1.3}$$

Because d is usually smaller, and sometimes significantly smaller than N, and because in our work we keep d fixed at 12 coefficients we put our algorithm in big-O notation in terms of N_Z. The predominating term comes from the multiplication of $N_Z - 2d - 2) \times K_{var}^{Z}$. Now $N_Z - 2d - 2$ is linear in N_Z, as is K_{var}^{Z}. Therefore the total cost is lower bounded as:

$$K_{OCD-Chen} = O(N_Z^2) \tag{1.4}$$

This power of two is the source of the slowness of DACDec2, which repeatedly apply OCD-Chen to detect multiple changepoints. A faster method is possible.

2. Recursive OCD-Chen

The main source of computational cost in the preceding algorithm is the computation of the variance for a sequence of vectors or matrices, each successive vector differing from the preceding only by the inclusion or exclusion of one data vector. This method of calculation is wasteful and may be better done in a recursive fashion, wherein after a computation of the variance of the initial $X_{i=d+1}$ and $Y_{i=d+1}$, we update the variance. To compute the subsequent $\sigma^2_{X_{i=d+2}}$, we update $\sigma^2_{X_{i=d+1}}$, this update takes into account the inclusion of the data vector $Z(d+2,:)$ (X grows by one vector.) To compute $\sigma_{Y_{i=d+2}}$, we update $\sigma_{Y_{i=d+1}}$ to take into account the exclusion of the same vector $Z(d+2,:)$ (Y shrinks by one vector.) These updates are peformed as in the following subsections.

2.1. Update of means

The usual way to compute a mean is:

$$\mu_l = \frac{1}{l} \sum_{j=1}^{l} x(j,:) \tag{2.1}$$

with l being the last index of the segment $x(1:l,:)$.

The formula for updating the mean to include a new vector is simple.

$$\mu_{l+1} = \frac{l \times \mu_l + Z_l}{l+1} \tag{2.2}$$

We essentially convert the old mean to a sum by multiplying it by the number of elements used to compute it, then add to this sum and divide by the new length. This is what is done to compute μ_{X_i} where

$$l = N_{X_i} = \text{length}(Z(1:i,:)) = i. \tag{2.3}$$

This give us:

$$\mu_{X_{i+1}} = \frac{i \times \mu_{X_i} + Z(i+1,:)}{i+1} \tag{2.4}$$

For updating the mean to exclude a vector, we have

$$\mu_{l-1} = \frac{l \times \mu_l - Z_l}{l-1} \tag{2.5}$$

To adapt this to the case of Y, using the knowledge that

$$l = N_{Y_i} = \text{length}(Z((i+1):N_Z,:)) = N_Z - i \tag{2.6}$$

we can derive:

$$\mu_{Y_{i+1}} = \frac{(N_Z - i) \times \mu_{Y_i} - Z(i+1,:)}{N_Z - i - 1} \tag{2.7}$$

2.2. Update of variances

The derivation of the formula for the update of the variances is more involved. Here we make the equivalence $Z_j \equiv Z(j,:)$. Normally, the unbiased estimator of the variance is computed for any given data as

$$\sigma_l^2 = \frac{1}{l-1}\sum_{j=1}^{l}(Z_j - \mu_l)^2 \tag{2.8}$$

Subtracting one from the length we have

$$\sigma_l^2 = \frac{1}{l-2}\sum_{j=1}^{l-1}(Z_j - \mu_{l-1})^2 \tag{2.9}$$

which is equivalent to:

$$(l-2)\times\sigma_l^2 = \sum_{j=1}^{l-1}(Z_j - \mu_{l-1})^2 \tag{2.10}$$

Taking the last term of the sum in equation 2.8 out of the sum:

$$\sigma_l^2 = \frac{(Z_l - \mu_l)^2}{l-1} + \frac{1}{l-1}\sum_{j=1}^{l-1}(Z_j - \mu_l)^2 \tag{2.11}$$

Further, we make use of the following identity

$$\begin{aligned}
&\sum_{j=1}^{l-1}(Z_j - \mu_l)^2 - \sum_{j=1}^{l-1}(Z_j - \mu_{l-1})^2 = \\
&-2\sum_{j=1}^{l-1} Z_j\mu_l + \sum_{j=1}^{l-1}\mu_l^2 + 2\sum_{j=1}^{l-1} Z_j\mu_{l-1} - \sum_{j=1}^{l-1}\mu_{j-1}^2 \\
&= -2(l-1)\mu_l\mu_{l-1} + (l-1)\mu_l^2 + 2(l-1)\mu_{l-1}^2 - (l-1)\mu_{l-1}^2 \\
&= (l-1)[\mu_l - \mu_{l-1}]^2
\end{aligned} \tag{2.12}$$

in order to restate the second term in equation 2.11:

$$\frac{1}{l-1}\sum_{j=1}^{l-1}(Z_j - \mu_l)^2 = \frac{1}{l-1}(l-1)\left[\mu_l - \mu_{l-1}\right]^2 + \frac{1}{l-1}\sum_{j=1}^{l-1}(Z_j - \mu_{l-1})^2 \tag{2.13}$$

This leads us to:

$$\sigma_l^2 = \frac{(Z_l - \mu_l)^2}{l-1} + (\mu_l - \mu_{l-1})^2 + \frac{1}{l-1}\sum_{j=1}^{l-1}(Z_j - \mu_{l-1})^2 \tag{2.14}$$

Let us expand the second term of equation 2.14, remembering equation 2.5.

$$\begin{aligned}
(\mu_l - \mu_{l-1})^2 &= \left(\mu_l - \frac{l\times u_l - Z_l}{l-1}\right)^2 \\
&= \left(\frac{1}{l-1}Z_l - \frac{1}{l-1}\mu_l\right)^2 \\
&= \frac{(Z_l - \mu_l)^2}{(l-1)^2}
\end{aligned} \tag{2.15}$$

Using equations 2.15 and 2.10 in equation 2.14, we have

$$\begin{aligned}
\sigma_l^2 &= \frac{(Z_l - \mu_l)^2}{l-1} + \frac{(Z_l - \mu_l)^2}{(l-1)^2} + \frac{l-2}{l-1}\sigma_{l-1}^2 \\
\sigma_l^2 &= \frac{l}{(l-1)^2}(Z_l - \mu_l)^2 + \frac{l-2}{l-1}\sigma_{l-1}^2
\end{aligned} \tag{2.16}$$

which, substituting $l \equiv l+1$ finally yielding

$$\sigma_{l+1}^2 = \frac{l+1}{l^2}(Z_{l+1} - \mu_{l+1})^2 + \frac{l-1}{l}\sigma_l^2 \tag{2.17}$$

Conversely if we are considering the removal of a vector, we can derive from equation 2.16 the following:

$$\begin{aligned}\sigma_{l-1}^2 &= \frac{l-1}{l-2}\left(\sigma_l^2 - \frac{l}{(l-1)^2}(Z_l - \mu_l)^2\right) \\ &= \frac{l-1}{l-2}\sigma_l^2 - \frac{l}{(l-1)(l-2)}(Z_l - \mu_l)^2 \end{aligned} \tag{2.18}$$

Now we can more directly translate equations 2.17 and 2.18 to our needs for the vectors X_i and Y_i. Equation 2.17 becomes, with the relation given in equation 2.3,

$$\sigma_{X_{i+1}}^2 = \frac{i+1}{i}\left(Z(i+1,:) - \mu_{X_{i+1}}\right)^2 + \frac{i-1}{i}\sigma_{X_i}^2 \tag{2.19}$$

whereas equation 2.18 becomes, given the relation in equation 2.6,

$$\sigma_{Yi+1}^2 = \frac{N_Z - i - 1}{N_Z - i - 2}\sigma_{Y_i}^2 - \frac{N_Z - i}{(N_Z - i - 1)(N_Z - i - 2)}(Z(i+1,:) - \mu_{Y_{i+1}})^2 \tag{2.20}$$

2.3. Practical Implementation

We begin with a standard computation of the means and variances of X_i and Y_i with $i = d + 1$. We then run a look where i increments from $i = d+2$ to $i = N_Z - d - 2$, and compute the updates of the mean and variance.

Algorithm 4 Recursive OCD-Chen

(1) $N_Z = \text{length}(Z)$;
(2) $\sigma_Z^2 = \text{var}(Z)$;
(3) Take the determinant of the covariance in the diagonal case
(4) the product of the elements in the variance vector:
(5) $D_0 = \prod_{j=1}^{d} \sigma_{Z_i}^2(j)$;
(6) $\mu_X = \text{mean}(Z(1:(d+1),:))$;
(7) $\mu_Y = \text{mean}(Z((d+2):N_Z,:))$;
(8) $\sigma_X^2 = \text{var}(Z(1:(d+1),:))$;
(9) $\sigma_Y^2 = \text{var}(Z((d+2):N_Z,:))$;
(10) $L_X = d + 1$;
(11) $L_Y = N_Z - d - 1$;
(12) for $i = d + 2$ to $N_Z - d - 2$ do
(13) $\quad L'_X = L_X + 1$;
(14) $\quad L'_Y = L_Y - 1$;
(15) $\quad x = Z(i,:)$;
(16) $\quad \mu'_X = (\mu_X \times L_X + x)/(L'_X)$;
(17) $\quad \mu'_Y = (\mu_Y \times L_X - x)/(L'_Y)$;
(18) $\quad {\sigma_X^2}' = (L_X/{L_X}') \times \sigma_X + ((x - \mu'_X)^2)/L_X$;
(19) $\quad {\sigma_Y^2}' = (L_X/{L_X}') \times (\sigma_Y - (x - \mu'_Y)^2)/L'_X)$;
(20) $\quad D_1 = \prod_{j=1}^{d} \sigma_{X_i}^2(j)$
(21) $\quad D_2 = \prod_{j=1}^{d} \sigma_{Y_i}^2(j)$; ;
(22) $\quad \Delta BIC(i) = N_Z \log D_0 - i \log D_1 - (N_Z - i) \log D_2 - \lambda P$;
(23) $\quad L_X = {L_X}'$; Update the lengths, means, and variances
(24) $\quad L_Y = {L_Y}'$;
(25) $\quad \mu_X = {\mu_X}'$;
(26) $\quad \mu_Y = {\mu_Y}'$;
(27) $\quad \sigma_X^2 = {\sigma_X^2}'$;
(28) $\quad \sigma_Y^2 = {\sigma_Y^2}'$;
(29) end
(30) $i_{change} = \arg\max_i \Delta BIC(i)$

2.4. Computational Cost Analysis

We the same costs for most of the operations in the algorithm. We again ignore the cost of computing the length of Z. The cost of computing the variance of Z on line 2 is still K_{var}^Z, and the cost of computing the

products on lines 5, 20, and 21 remains K_p. Generally we have the following cost for the algorithm:

$$K_{var}^{Z} + K_p + K_{var}^{X} + K_{var}^{Y} + K_{loop} \tag{2.21}$$

The costs of computing the means on lines 6 and 7 is included in the cost of computing the variance of X and Y, using an equation equivalent to that of 1.1. In any case, as we saw in the standard OCD-Chen algorithm, the cost of the operations preceding the loop will not tend to dominate the cost of the algorithm. We therefore take a closer look at K_{loop}. Here we have the cost of two additions on line 13 and 14. Next to update the mean on line 16 requires $K_{ud1} = d \times (c_{mult} + c_{add} + c_{div})$, and the same, K_{ud2} for line 17. Updating the variance of X requires $K_{ud3} = 2 \times c_{div} + 3 \times d \times c_{mult} + d \times c_{add}$. Updating the variance carries the same cost. We again have two times K_p and $K_B IC$. We see that the cost of operations inside the loop does not depend on N_Z. Therefore the cost of the loop is linear in N_Z and on the whole the algorithm is of order

$$K_{Recursive} = O(N_Z) \tag{2.22}$$

Compare this with the standard algorithm which had a run time of order $O(N_Z^2)$ and we see that the recursive method gives a substantial savings in computational cost.

Appendix D.

Converting the Scores of Esposito

Esposito [1] is a frequently-cited source in phoneme segmentation. The performance scores for the author's algorithms are often referenced in works dealing with this subject. However, the authors do not compute the standard segmentation performance measures in the conventional way, leading to the appearance of an advantage. Here we detail the process of converting the performance scores found in [1] to their equivalents under the conventional way of computing the standard scores.

First, we note the scores we cite are those of section 4.3 of [1]. On page 278 it is stated that the results in this section are given for testing on the whole *Test* directory of TIMIT. The principal score given for each method is P_c, (corresponding to our HR.) On page 279, it is stated that this score for Melbank encoding is given "when the false alarm rate has a value of around 0.05, corresponding roughly to a $D = 0$ over-segmentation rate." This phrasing is repeated when stating the P_c score for other encoding schemes such as MFCC and Log-area ratios, as well as for other methods such as SVF.

The P_c scores for the method developed by the authors of [1] under the different encoding schemes are as follows:

Table D.1.: P_c given in Section 4.3 of [1] for their method

Encoding	P_c
Melbank	82%
MFCC	76%
Log-Area Ratios	70%

For SVF based on MFCCs, $P_c = 67\%$ was cited on page 280.

Based on this P_c and the constraint on False Alarm rate referenced above, we can derive our traditional scores. First some terminology. On page 269 [1] defines S_t as the overall number of phone boundaries contained in the prototype database. S_c is the number of correctly detected boundaries. Then

$$P_c = 100 \times \frac{S_c}{S_t} \tag{0.1}$$

which is equation 8 of [1]. This corresponds to our HR. However [1] diverges from our measures in the definition of FAR, or P_{fa} as it is called in that work. For our purposes, FAR is defined as 1 minus the ratio of the number of hits to the number of total detections, or as percentage

$$FAR = \frac{\text{detections} - \text{hits}}{\text{detection}} \times 100 \tag{0.2}$$

In [1], P_{fa} is defined as a fraction, with the numerator being the difference in the number of detections and hits $S_d - S_c$, and the denominator being S_{nt} which is equal to $S_f - S_t$.

$$P_{fa} = 100 \times \left(\frac{S_d - S_c}{S_{nt}} \right) \tag{0.3}$$

[1] calls S_{nt} the number of "non-target segmentation frames identified as targets" which could be called the number of speech frames not corresponding to a phoneme segmentation boundary that were identified as such. [1] S_f is the total number of speech frames contained in the speech corpus and S_t is the total number of boundaries contained in the reference transcription of the database.
We have the following equivalences with our methods: S_d = detections, S_c = hits, S_t = transcriptions

[1] However, this does not seem to be the appropriate meaning of this number. Remembering their definitions on this page, we see that S_f and S_t are fixed. Therefore it is puzzling to state that this number, $S_{nt} = S_f - S_t$ is associated with the erroneous identification of speech frames as segmentation boundaries, since it does not depend on the output of the algorithm at all.

(the number of reference transcriptions in the Test corpus.) Further, S_f is the number of speech frames in the Test corpus. [1] cites on page 275 that $S_f = 516165$ and $S_t = 62465$.

We can compute the raw number of hits as

$$\text{hits} = S_c = P_c \times S_t \tag{0.4}$$

and the raw number of detections as

$$\text{detections} = P_{fa} \times (S_f - S_t) + S_c = P_{fa} \times (S_f - S_t) + P_c \times S_t \tag{0.5}$$

With transcriptions S_t already being given we can calculate our partial performance measures as usual:

$$HR = \frac{hits}{transcriptions} = \frac{P_c \times S_t}{S_t} \tag{0.6}$$

$$OS = \frac{detections}{transcriptions} - 1 = \frac{P_{fa} \times (S_f - S_t) + P_c \times S_t}{S_t} \tag{0.7}$$

$$FAR = 1 - \frac{hits}{detections} = 1 - \frac{P_c \times S_t}{P_{fa} \times (S_f - S_t) + P_c \times S_t} \tag{0.8}$$

with

$$F1 = \frac{2 \times PCR \times HR}{PCR + HR} \tag{0.9}$$

and

$$r_1 = \sqrt{(100 - HR)^2 + OS^2} \tag{0.10}$$

$$r_2 = \frac{-OS + HR - 100}{\sqrt{2}} \tag{0.11}$$

$$R = 1 - \frac{abs(r_1) + abs(r_2)}{200} \tag{0.12}$$

It must also be kept in mind the method by which the best scores of the algorithm [1] were reported. Since the authors did not have R-value in mind as a performance criterion, their algorithm was not tuned to maximize it. Namely, the authors chose a configuration which, according to their measures, gave a FA of 5% which they claim corresponds to an oversegmentation of "roughly 0".

Bibliography

[1] A. Esposito and G. Aversano, "Text independent methods for speech segmentation," in *Lecture Notes in Computer Science: Nonlinear Speech Modeling* (C. G. et al et al, ed.), pp. 261–290, Springer Verlag, 2005.

[2] S. Dusan and L. Rabiner, "On the relation between maximum spectral transition positions and phone boundaries," in *INTERSPEECH 2006*, pp. 645–648, Sept. 2006.

[3] D. T. Toledando, Luis A. Hernández Gómez, and L. V. Grande, "Automatic phonetic segmentation," *IEEE Transactions on Speech and Audio Processing*, vol. 11, pp. 617–625, November 2003.

[4] O. J. Räsänen, U. K. Laine, and T. Altosaar, "Blind segmentation of speech using non-linear filtering methods," in *Speech Technologies* (I. Ipsic, ed.), InTech, 2011.

[5] P. Delacourt and C. J. Wellekens, "DISTBIC: A speaker-based segmentation for audio data indexing," pp. 111–126, 2000.

[6] M. Sharma and R. J. Mammone, ""blind" speech segmentation: Automatic segmentation of speech without linguistic knowledge," in *Spoken Language Processing, 4th International Conference on*, pp. 1237–1240, ISCA, October 1996.

[7] M. Artimy, W. Robertson, and W. Phillips, "Automatic detection of acoustic sub-word boundaries for single digit recognition," in *Electrical and Computer Engineering, 1999 IEEE Canadian Conference on*, vol. 2, pp. 751–754 vol.2, 1999.

[8] C. Mitchell, M. Harper, and L. Jamieson, "Using explicit segmentation to improve hmm phone recognition," in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 1, pp. 229–232 vol.1, may 1995.

[9] S. R. Quackenbush, T. P. Barnwell, and M. A. Clements, *Objective Measures of Speech Quality*. Prentice Hall, 1988.

[10] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "DARPA TIMIT acoustic phonetic continuous speech corpus CDROM," 1993.

[11] M. Brookes, "VOICEBOX: Speech processing toolbox for MATLAB," http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html.

[12] G. Almpanidis and C. Kotropoulos, "Phonemic segmentation using the generalised gamma distribution and small sample bayesian information criterion," *Speech Communication*, vol. 50, pp. 38–55, January 2008.

[13] O. J. Räsänen, U. K. Laine, and T. Altosaar, "An improved speech segmentation quality measure: the r-value," *Proceedings of the 10th Annual Conference of the International Speech Communication Association 2009 Interspeech 09*, no. 5, pp. 1851–1854, 2009.

[14] S.-S. Cheng, H.-M. Wang, and H.-C. Fu, "BIC-based speaker segmentation using divide-and-conquer strategies with application to speaker diarization," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 18, pp. 141–157, jan. 2010.

[15] S. Chen and P. S. Gopalakrishnan, "Speaker, environment and channel change detection and clustering via the bayesian information criterion," in *DARPA Broadcast News Transcription and Understanding Workshop*, pp. 127–132, February 1998.

[16] M. D. Skowronski and J. G. Harris, "Exploiting independent filter bandwidth of human factor cepstral coefficients in automatic speech recognition," *Journal of the Acoustical Society of America*, 2004.

[17] M. D. Skowronski and J. G. Harris, "Human factor cepstral coefficients," *Journal of the Acoustical Society of America*, 2002.

[18] B. J. Moore and B. R. Glasberg, "Suggested formula for calculating auditory-filter bandwidth and excitation patterns," *Journal of the Acoustical Society of America*, 1983.

[19] I. Mporas, T. Ganchev, and N. Fakotakis, "Speech segmentation using regression fusion of boundary predictions," *Comput. Speech Lang.*, vol. 24, pp. 273–288, April 2010.

[20] "Home page of mark d. skowronski."

[21] V. Khanagha, K. Daoudi, O. Pont, and H. Yahia, "Improving text-independent phonetic segmentation based on the microcanonical multiscale formalism," May 2011.

[22] G. Aversano, A. Esposito, and M. Mariano, "A new text-independent method for phoneme segmentation," in *Proceedings of the IEEE International Workshop on Circuits and Systems*, Aug. 2001.

[23] A. Kondoz, *Digital Speech: Coding for Low Bit Rate Communication Systems*, ch. Voice Activity Detection, pp. 357–377. John Wiley & Sons, 2004.

[24] J. Sohn, S. Member, N. S. Kim, and W. Sung, "A statistical model-based voice activity detection," *IEEE Signal Process. Lett*, vol. 6, pp. 1–3, 1999.

[25] D. Ying, Y. Yonghong, D. Jianwu, and F. K. Soong, "Voice activity detection based on an unsupervised learning framework," *Audio, Speech, and Language Processing, IEEE Transactions on*, 2011.

[26] M. Huijbregts and F. de Jong, "Robust speech/non-speech classification in heterogeneous multimedia content," *Speech Communication*, 2010.

[27] J. Ramírez, M. Gorríz, and J. C. Segura, *Robust Speech Recognition and Understanding*, ch. Voice Activity Detection. Fundamentals and Speech Recognition System Robustness. 2007.

[28] G. Almpanidis, M. Kotti, and C. Kotropoulos, "Robust detection of phone boundaries using model selection criteria with few observations," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 17, pp. 287–298, feb. 2009.

[29] M. Kotti, E. Benetos, and C. Kotropoulos, "Computationally efficient and robust bic-based speaker segmentation," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, pp. 920–933, july 2008.