InterACT
International center
for Advanced Communication Technologies
Universität Karlsruhe (TH)
Carnegie Mellon University
Prof. Dr. rer. nat. Alex Waibel

A Comparative Study of Gaussian Selection Methods in Large Vocabulary Continuous Speech Recognition

Studienarbeit

Author: Dirk Gehrig

Advisors: Carnegie Mellon University, Dr.-Ing. Thomas Schaaf
Universität Karlsruhe (TH), Dipl.-Inform. Sebastian Stüker

Hiermit versichere ich, diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Pittsburgh, den 31. März 2006

[illegible]

Contents

Chapter 1

Fast and Accurate Speech Recognition

In this first chapter I give a short motivation for speech recognition and also a brief introduction. I explain, that it is desirable for a lot of the state-of-the-art systems to be large and that the speed of this speech recognizer can be a problem. It can be necessary, to speed it up, but it is desirable to do this without loss in output quality. Afterwards I describe the basic idea for speeding up a speech recognizer, which was addressed in this thesis. This is Gaussian selection.

1.1 Introduction to Speech Recognition

At all times applications were build, which tried to make life simpler. One of the recent approaches is to use speech recognition and speech synthesis for the interaction with computers and embedded systems. Some systems try to simplify the useability of already invented techniques like cell phones or navigation systems, while others open up new possibilities like intelligent rooms that react on the situation inside the room. This can partly be done using speech recognition. Together with machine translation it opens up even more perspectives, e. g. the simultaneous translation during a phone call between two people that are not able to speak the languages of each other.

Speech recognition transforms the audio data of each speaker in a transcription of what the speaker said. The first step is to extract specific features out of the audio data. Having the features most of the state-of-the-art systems use hidden Markov models (HMMs) [Rab90] to get a probability for each possible word given the feature vectors belonging to a series of time stamps. Most HMM-systems model the probability density functions of the HMM-states as Gaussian mixtures. This Gaussian mixtures have to be evaluated, when the probability of a certain HMM-state is needed. Using a language model the most likely sentences are calculated out of the probabilities of the possible words.

1.2 Time-expensive Evaluation of Gaussian Mixtures

For some recognition tasks a relatively small number of HMM-states might be sufficient, but with a increasing number of words the recognizer should be able to recognize, the number of HMM-states has to be increased to gain the same accuracy. Since the pronunciation of a phoneme can vary for different speakers and contexts, the accuracy can be further improved by using more than one HMM-state sequence for each word. Therefore with a greater number of states the accuracy of the system can be improved. This can only be done, since more and more training data is available nowadays. Another advantage of more training data is the possibility to build fully continuous speech recognizers, what means, that every HMM-state has a unique Gaussian mixture model.

Due to this reasons most of the state-of-the-art speech recognizers use some thousand HMM-states. For fully continuous systems the number of Gaussian mixtures that have to be evaluated for each time frame during the recognition increases in the same manner as the number of HMM-states. The evaluation of thousands of this Gaussian mixtures (see appendix A.1) needs a lot of computation time. In some applications only the accuracy of the speech recognizer is important and the time it needs for the recognition is unimportant, but there are a lot of systems in which the time behavior is also of fundamental importance. If the recognizer is part of a system, that interacts with a human user, it is important that the recognizer is fast, since it is inconvenient for the user to wait for the system and therefore would not be of great help.

1.3 Gaussian Selection

It is possible to speed up speech recognizers in different ways. One common way is to consider only those states that are most likely given the previous states. This approach restricts the search space for the best transcription and therefore speeds up the recognition task. When evaluating the Gaussian mixtures of a HMM-state at certain coordinates, it seems reasonable that only one or a few Gaussians have significant contributions to the probability of the Gaussian mixture. This are the Gaussians, that have a significant probability at the position of the feature vector, i. e. mostly the Gaussians that are close to the feature vector. If the feature vector lies on the tail of a Gaussian it is not necessary to evaluate this Gaussian, since it would not have a significant contribution to the overall probability of the Gaussian mixture.

When a speed up of the speech recognizer should be achieved by only using the most significant Gaussians for each pair of feature vector and Gaussian mixture, it becomes an important task of the speech recognizer to find this significant Gaussians. To achieve an overall speed up of the recognizer the search for the significant

Gaussians has to be fast, but the selected Gaussians should also give a good approximation of the Gaussian mixture. Therefore different Gaussian selection methods have been proposed in recent years. Some methods use a segmentation of the feature space to define different areas, whereas only the Gaussians of one or a few areas are used for the calculation of a Gaussian mixture during recognition. The areas, that have to be evaluated, are chosen depending on the feature vector. This methods are multidimensional feature space partitioning trees and vector quantization. Other methods do not use any preprocessing, but determine the most significant Gaussians using distance measures, that are faster in calculation than evaluating the Gaussian. This methods were proposed as projection search and Hamming-distance-approximation.

Since each of the different methods has parameters that have to be tuned before using the Gaussian selection method, it is difficult to choose the best Gaussian selection method and find the best parameters for a specific task. The proposed methods can differ in the computation time and the memory, that is needed to calculate the most significant Gaussians, but they can also differ in the quality of the chosen Gaussians. In addition to that, they vary in their useability when speaker adaptation is performed during the speech recognition. This thesis will give an overview on the behavior of the most promising methods compared to a baseline system. It considers effects of different parameter settings and differences between the methods.

Chapter 2

Gaussian Selection

In this chapter I give an overview of the four different Gaussian seleciton categories. I describe four representative methods of Gaussian selection one for each of the four categories. This methods are the bucket box intersection algorithm, vector quantization, projection search and Hamming-distance-approximation.

2.1 Classification of Methods

Different methods have been proposed to select the subset of Gaussians that should be used to calculate an approximated probability for a Gaussian mixture. Ortmanns [Ort98] suggests to separate them into four different categories, which are k-dimensional search trees, vector quantization, projection search and Hamming-distance-approximation. The first two are based on a static partitioning of the feature space. Therefore the feature space is partitioned in appropriate cells, which can be generated before the evaluation. During evaluation only the Gaussians of the cells, that are closest to the observation vector, are calculated to get an approximation of the Gaussian mixture. If only the closest cell is used, then it is the cell, in which the observation vector is located. Using projection search or Hamming-distance-approximation the selection of the Gaussians, that have to be calculated, is done by roughly approximating the distances between the observation vector and the Gaussians during evaluation.

2.2 BBI

The bucket box intersection (BBI) algorithm [FR96, WF97] is an approach to approximate Gaussian mixtures with diagonal covariance matrices, based on a binary feature space partitioning tree [Ben75]. For a given feature vector the tree is used to find the set of the most significant Gaussians for this vector. The basic structure used in the BBI algorithm is the k-dimensional search tree, which partitions the k-dimensional feature space in disjoint rectangular subspaces. Each leaf of the tree represents one of the rectangular subspaces, that are described by hyperplanes defined in the inner nodes. Each hyperplane is orthogonal to one of the coordinate axis. The hyperplane belonging to the root node divides the feature space in two half spaces. Depending on the location of these sub-spaces in relation to the hyperplane one is the left and one the right child node. Every child node is further divided into two child nodes. This is repeated until the tree has a depth of d . Each Gaussian is stored in lists (bucket) belonging to the sub-spaces in which the value of the Gaussian is above a certain threshold. After this precalculation it is possible to find the subspace in which a Gaussian is located by d comparisons. In every node it has to be decided on which side of the hyperplane the observation vector is located and it has to be descended to the appropriate child. When a leaf is reached only the Gaussians in this leaf are used to calculate an approximation of the Gaussian mixture probability.

The log of a single multivariate Gaussian probability density function with diagonal covariances, e. g. $\Sigma = \mathbf{I}\sigma$ with $\sigma = (\sigma_1^2, \ldots, \sigma_K^2)^T$, can be calculated as:

$$logN(x, \mu, \Sigma) = -\frac{1}{2}[log((2\pi)^K \prod_{j=1}^{K} \sigma_j^2) + \sum_{j=1}^{K} \frac{(x_j - \mu_j)^2}{\sigma_j^2}] \tag{2.1}$$

For a given threshold R the region of the Gaussian with higher probabilities than $R \cdot N(\mu, \mu, \sigma)$, i. e. $N(x, \mu, \sigma) > R \cdot N(\mu, \mu, \sigma)$, is a hyperellipsoid with axis parallel to the coordinate axis. A box with boundary hyperplanes parallel to the coordinate axis can be calculated, so that it completely includes the hyperellipsoid. Fritsch and Rogina call it the Gaussian box associated with R (see figure 2.1).

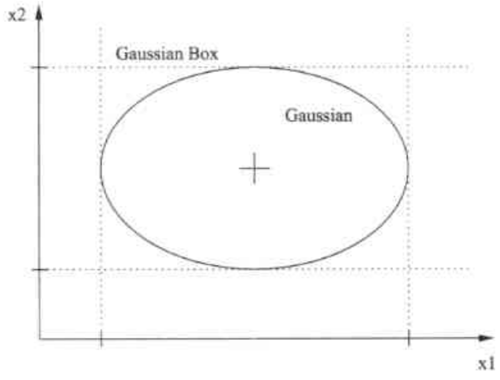


Figure 2.1: Box around a Gaussian, that includes the part above a certain threshold

The interval $[a_j, b_j]$ of a Gaussian box to coordinate j for $0 \leq R \leq 1$ is given by:

$$[a_j, b_j] = \mu_j \pm \sqrt{-2\sigma_j^2 log(R)} \tag{2.2}$$

To find the most significant Gaussian boxes for a feature vector, a K-dimensional space partitioning tree is used. The standard BBI algorithm [FR96], as described by Fritsch and Rogina, creates one tree for every single codebook. At every nonterminal node of the tree the region belonging to this node is divided into two half-spaces by a hyperplane orthogonal to one of the coordinate axis. To find locally optimized hyperplanes, three steps are processed:

1. Identify the Gaussian boxes intersecting with the current hyperrectangular region corresponding to the node.
2. For all coordinate axis x_j sort the L(ower) and U(pper) boundaries of the selected Gaussian boxes. Hypothesize a division hyperplane, so that it has an equal number of L boundaries on the left side and R boundaries on the right side. This is to create a balanced tree, that has a similar number of Gaussian boxes in each leaf node. Calculate the number C_i of selected Gaussian boxes that are split by the hyperplane.
3. Select the hyperplane with the minimum number of splits C_i as the current nodes division hyperplane. If there are two or more hyperplanes with the same number of splits, an additional rule has to be used, e. g. select the hyperplane belonging to the coordinate with the lowest index.

During speech recognition the binary trees can be used to find the leafs with the most significant Gaussian boxes concerning a given feature vector. This is done by traversing the tree top down. In every node the decision has to be made on which side of the hyperplane the feature vector is located by evaluating the coordinate orthogonal to the hyperplane. Knowing the side of the hyperplane on which the feature vector is located the appropriate child node can be selected. When a leaf node is reached, only the Gaussians belonging to the Gaussian boxes in this leaf node are evaluated to get an approximation of the log probability of the Gaussian mixture.

Using separate trees for every codebook needs more calculation time than one single tree for all codebooks, because every tree has to be evaluated separately. Therefore Fritsch and Woszczyna have proposed the big-BBI algorithm [WF97], which calculates a single binary tree for all codebooks. Using one tree for all codebooks the leaf belonging to a feature vector does not need to contain Gaussian boxes of the codebook, that has to be evaluated. For this kind of codebook and leaf pairs back-off vectors have to be calculated, i. e. for every codebook with no Gaussian box in a leaf the Gaussian box that is nearest to the leaf has to be found and added to the leafs list of Gaussian boxes .

Using a threshold of R for the Gaussian boxes, the maximum error that can be achieved during evaluation of a single Gaussian is $R \cdot N(\mu, \mu, \sigma)$. Evaluating the Gaussian mixture $p(x|w)$ with diagonal covariance matrices for class ω as

$$p(x|w) = \sum_{i=1}^{N} c_{\omega i} N(x, \mu_{\omega i}, \sigma_{\omega i}) \tag{2.3}$$

with the vector of mixture coefficients $(c_{\omega 1}, \dots, c_{\omega N})$ for class ω satisfying the contraints $c_{\omega j} \geq 0$ and $\sum_{j=1}^{N} c_{\omega j} = 1$, by restricting the evaluation to the Gaussians with Gaussian boxes that contain the feature vector, the overall error is smaller than $R \cdot \sum_{i=1}^{N} c_{\omega i} N(\mu_{\omega i}, \mu_{\omega i}, \sigma_{\omega i})$. Fritsch and Rogina have also investigated absolute thresholds, but have examined that they get better results with relative thresholds. The two parameters that have to be tuned, when using the BBI as Gaussian selection, are the depth of the tree and the value of the relative threshold .

2.3 Vector Quantization (Clustering)

To select the Gaussians with a significant contribution to the probability of a Gaussian mixture vector quantization can be used. Therefore the feature space has to be partitioned in cells before the decoding, whereas a centroid is calculated for each cell. During evaluation only the Gaussians of one or a few cells with centroids closest to the feature vector are evaluated. This method was first published by Boccherie [Boc93], but a lot of systems use slightly different methods [GKY99, SPZ05, HS97, SHH+99, WSTI95, AM01]. Some of them are described later.

In almost every system for the calculation of the cells a k-means like algorithm described in appendix A.3 is used. In many systems a weighted Euclidean distance

$$d(\mu_{\omega_1}, \mu_{\omega_2}) = \frac{1}{K} \sum_{j=1}^{K} w_j (\mu_{\omega_1 j} - \mu_{\omega_2 j})^2 \tag{2.4}$$

is used as distance metric for the k-means algorithm [Boc93, GKY99, HS97, SHH+99]. Other distance measures can be used as well, e. g. Euclidean distance or a simplified form of the symmetric Kullback-Leibler divergence [SPZ05], which has the following form:

$$d(\mu_{\omega_1}, \mu_{\omega_2}) = \sum_{j=1}^{K} \left(\frac{\sigma_{\omega_1 j}^2 + (\mu_{\omega_1 j} - \mu_{\omega_2 j})^2}{\sigma_{\omega_2 j}^2} + \frac{\sigma_{\omega_2 j}^2 + (\mu_{\omega_1 j} - \mu_{\omega_2 j})^2}{\sigma_{\omega_1 j}^2} \right) \tag{2.5}$$

The Gaussians that are located on the border of one of the disjoint cells, could also have significant contributions to cells which they are not assigned to. Therefore after the clustering most system use additional rules to define, which Gaussians belong to what centroid. Mostly a Gaussian can be assigned to more than one cell. One simple case is to define a threshold $\Theta >> 1$ and assign every Gaussian to every centroid, which it is closer to than the threshold concerning a distance metric, e. g. $\frac{1}{K} \sum_{j=1}^{K} \frac{(c_{ij} - \mu_{\omega j})^2}{\sigma_{\omega j}^2} \leq \Theta$. But also more complex rules are used to define the final clusters [Boc93, GKY99, HS97]. It can happen that no Gaussian of a certain Gaussian mixture is assigned to a centroid. This results in the problem, that it is not possible to approximate the probability of this Gaussian mixture based on that cell. If only cells are selected for the evaluation, that do not contain any Gaussian of the Gaussian mixture, the probability can not be approximated. This can be solved by defining that every cell has to have at least one Gaussian of each Gaussian mixture or by state flooring, which means that the value for cells without a Gaussian of a certain Gaussian mixture is given by a fixed estimated value.

During recognition the one or more cells that are closest to the observation vector have to be determined. This can be done for example by calculating the distance between all centroids and the observation vector. Then the Gaussians of the closest cells are evaluated to get an approximated probability for the Gaussian mixture.

2.4 Projection Search

The projection search [NN96] is based on a dynamic partitioning of the feature space. The Gaussians, that are located in a rectangular subspace of size 2ϵ around the observation vector, are determined during decoding without any precalculation. Only this Gaussian are evaluated to get an approximated probability of the Gaussian mixture.

To find the Gaussians which are located in the rectangular subspace the feature space is bounded by two parallel hyperplanes orthogonal to the first coordinate axis. This hyperplanes are located with a distance of ϵ to the feature vector on both sides. The Gaussians between this two hyperplanes are stored in a candidates list for further use. Then the constructed subspace is bounded by two more hyperplanes which also have a distance of ϵ to the feature vector and which are orthogonal to the first ones and a coordinate axis. Only the Gaussians that are between this two hyperplanes are kept in the candidates list. The bounding of the subspace is progressed until a given number of coordinate axis is bounded by hyperplanes. The candidates list then contains all Gaussians that have to be evaluated to get an approximated value of the Gaussian mixture. The two parameters that specify the subspace and therefore the list of Gaussians, that have to be evaluated, are the distance between the hyperplanes and the feature vector and the number of coordinate axis that are bounded. If no Gaussian of a certain Gaussian mixture is located in the rectangular subspace state-flooring, i. e. a constant value, or a back-off Gaussian has to be used.

2.5 Hamming-Distance-Approximation

Another approach of Gaussian selection is the Hamming-distance-approximation [BU95]. It selects the most significant Gaussians by approximating the distance of the l_p-norm between the feature vector and each Gaussian. The approximation of the distances needs a lot less computation time than the evaluation of the Gaussians. For the approximation of the Gaussian mixture the n Gaussians with the smallest distances to the feature vector are used. Using the l_1-norm, the distance $d(x, y)$ between the feature vector x and the mean vector y of a Gaussian can be calculated as:

$$\begin{aligned} d(x,y) &= \sum_{k=1}^{K} |x_k - y_k| && (2.6) \\ &= \sum_{k=1}^{K} \left[|x_k| - |y_k| - \left\{ \begin{array}{ll} 2min(|x_d|, |y_d|) & , \; x_d \times y_d > 0 \\ 0 & , \; else \end{array} \right\} \right] && (2.7) \\ &= ||x||_1 + ||y||_1 - 2 \times \sum_{x_d \times y_d > 0} min(|x_d|, |y_d|). && (2.8) \end{aligned}$$

The Hamming-distance-approximation tries to calculate this distances efficiently by substituting the term $2 \times \sum_{x_d \times y_d > 0} min(|x_d|, |y_d|)$ with the value of the average vector component, i. e. $min(|x_d|, |y_d|)$ is substituted by $min\left(\frac{||x||_1}{K}, \frac{||y||_1}{K}\right)$.

Therefore the Hamming-distance $d_{HDA}(x, y)$ is:

$$d_{HDA}(x,y) = ||x||_1 + ||y||_1 - \frac{2}{K} min(||x||_1, ||y||_1) \times \sum_{x_d \times y_d > 0} 1 \qquad (2.9)$$

The advantage of this approximation lies in the simple calculation of the term $\sum_{x_d \times y_d > 0} 1$. Although the calculation of the approximation is fast, Ortmanns [Ort98] has experienced that the overall speed up of the speech recognizer is less than with projection search for the same WER. Therefore the Gaussian selection with Hamming-distance-approximation is not further investigated in this thesis.

Chapter 3

Experiments

In this chapter I first describe the measurements, which I used to investigate the different Gaussian selection methods. Then I give an introduction to the speech recognizer, that I used as a baseline system for my experiments. Afterwards I describe my investigation of the Gaussian seleciton methods bucket box intersection, projection search and clustering and explain the results of the according experiments. Finally I compare the results of the different Gaussian selection methods.

3.1 Measurements

For the evaluation of the various methods of Gaussian selection different measurement categories are used. The most important one is the word error rate (WER). The word error rate is a measurement for the quality of the speech recognizers output. It measures the number of wrong words in the output in relationship to the number of words of the optimal output. The error types wrong words can belong to are Insertion, Deletion and Substitution, which mean that a whole word has been inserted, deleted or substituted by another word. The word error rate is calculated using the sclite program, which is part of NIST's Speech Recognition Scoring Toolkit [nis]. Since no significance test were performed, it is assumed, that a system is as good as another system if the difference in the word error rate is 0.4% or smaller.

To measure the time behavior of a speech recognizer the average percentage of Gaussians per Frame, that are selected by the Gaussian selection methods, is measured. This is the average percentage of Gaussians per Frame, that have to be evaluated by the scoring function to get a probability for the Gaussian mixture. The scoring function used with the speech recognizer in this work only returns the probability of the Gaussian with the smallest mahalanobis distance to the feature vector for each Gaussian mixture. If the partly calculated distance of a Gaussian already exceeds the minimum distance, the calculation of this Gaussian is aborted and the probability is not calculated completely. The evaluated number of Gaussians per frame is also a measure for the Quality of the selected Gaussians, as it shows how many Gaussians are needed to get a certain performance.

Since the overhead for finding the nearest Gaussians is not taken into account, when just measuring the number of evaluated Gaussians per frame the real-time factor for the run of the speech recognizer is measured too. For this purpose the time that is needed by the speech recognizer for the recognition is measured without the start-up of the speech recognizer. To get the real-time factor this time is divided by the duration of the segmented audio data.

For the evaluation of the memory usage of the speech recognizer in the main memory and on the hard disk the size of the according data structures are measured.

3.2 Baseline System

For the discussion of the experiments it is assumed, that the reader is familiar with speech recognition and therefore only the parts, that are relevant for this work are explained in more detail. The speech recognizer that was used for the experiments is based on the JANUS Recognition Toolkit (JRTk) [WFK+96] developed at the University of Karlsruhe (TH) and the Carnegie Mellon University in Pittsburgh. The toolkit is an object-oriented platform for building state-of-the-art mutimodal recognizers. The components of a recognizer, which are objects, can be administrated using a Tcl/Tk script based environment.

The baseline system (see figure 3.1) used in the experiments is a fully continuous speech recognizer for modern standard Arabic speech in the broadcast news domain. It is based on HMMs and consists of 3112 states that represent 41 basic phonemes. Each phone consists of three states. The probabilities of the states are modeled by Gaussian mixtures. For the training of the Gaussian mixtures 85 hours of audio data were used. The first step in the preprocessing of the audio data is the segmentation of the audio data in speech and non-speech parts. Afterwards a automatic speaker clustering is performed, which tries to cluster data of each speaker in a certain environment based only on the audio data. The audio data is then transformed in 13 Mel-frequency cepstral coefficients (MFCC) [HH01] for each time-frame with a frame shift of $10ms$. The means and variances of each cluster are normalized afterwards. Using the MFCCs of the 7 frames before and after the current time-frame 195-dimensional feature vectors are created. Features that have little or no discriminative power should be removed, since they only increase the computational load and number of model parameters to be estimated without improving performance. To reduce the dimensionality of the feature space to a small number of dimensions while keeping the most important informations for a good separability Linear Discriminate Analysis (LDA) [Fuk90] is performed. The dimensions of the feature space are sorted according to their separability after the LDA. Therefore only the first 42 dimensions of the transformed feature space are used in the speech recognizer. To model the covariances of the Gaussians in the HMM-states diagonal covariance matrices are used. Therefore it is necessary to decorrelate the dimensions of the feature space to satisfy this constraint. The decorrelation of the dimensions during the LDA works best, if the training vectors are normal distributed and all Gaussians in a class have the same covariance. Since this is not the case for speech a single semi-tied covariance matrix [Gal99] is calculated to further decorrelate the elements of the feature vectors, but this also decreases the ordering of the dimensions according to their discriminative power. With this feature vectors the Gaussian mixtures were trained using a merge and split algorithm. Using the merge and split training about 160 000 Gaussians were obtained.

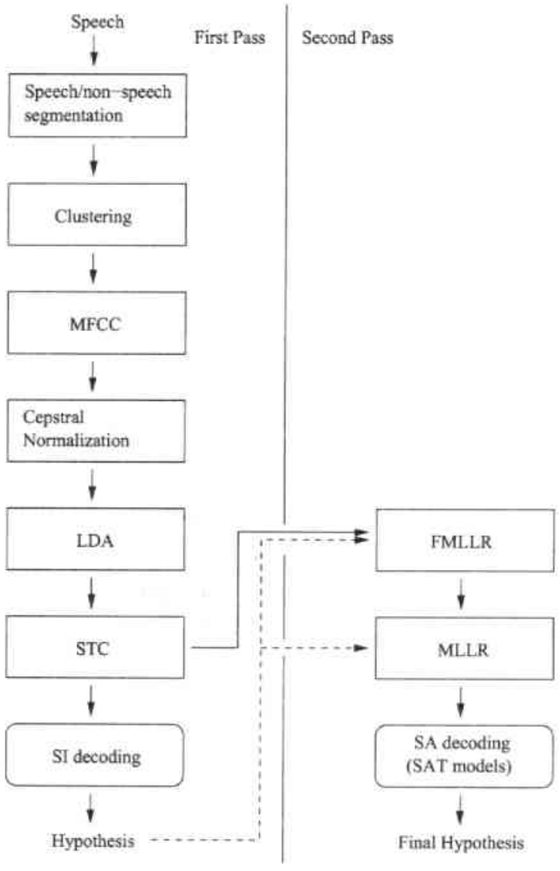


Figure 3.1: Basic structure of the baseline system

For the decoding, which is described in [SMFW01], two passes are used. The first pass is speaker independent and uses the described Gaussian mixtures. The second pass is speaker adaptive and therefore the Gaussian mixtures where further trained using feature-space maximum likelihood linear regression (FMLLR) [Gal97] for each cluster. During the speech recognition FMLLR and MLLR [Gal97, LW95] are used for speaker adaptation in the second pass based on the hypothesis of the first pass (see figure 3.1). The language model used for both passes has a perplexity of about 372. It uses about 19 million trigrams, 12 million bigrams and 115 thousand unigrams. The dictionary contains 115 000 words.

During the decoding, a list of acoustic models, that belong to the active states for a certain time frame, is passed to the scoring function (see figure 3.2). The scoring function then evaluates all Gaussians for each Gaussian mixtures, but returns only the value of the most important Gaussian, given an observation vector, for each Gaussian mixture. To speed up the search for the most important Gaussian the Gaussian selection is used. Therefore this work will focus only on this part of the speech recognizer.

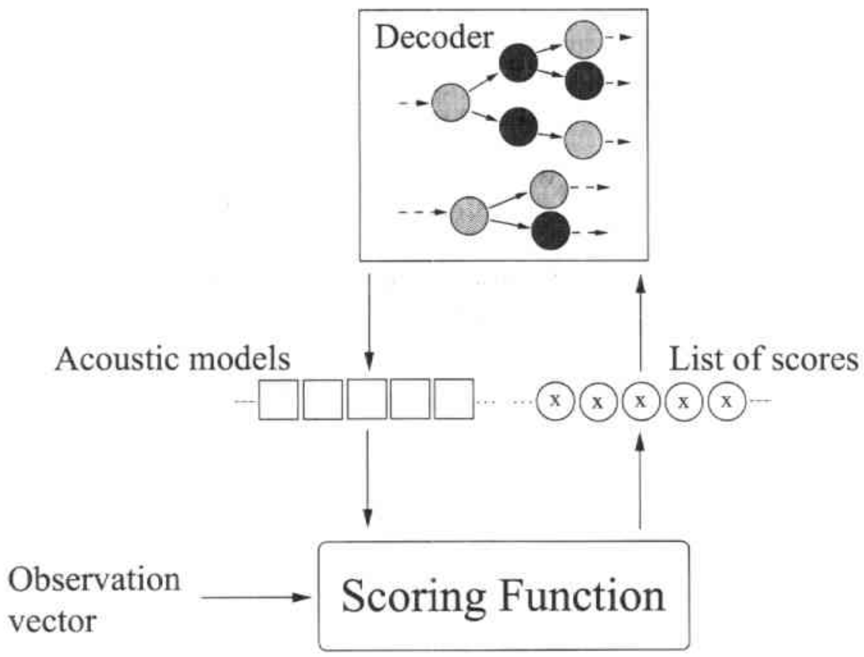


Figure 3.2: Interface between decoder and scoring function

For the experiments a test data set of about 1 hour and 15 minutes was used. The experiments that included time measurement were processed on Intel Pentium 4 computers with 3.2 GHz and 2 GByte of main memory. On this machines the baseline system needed 4.4 times real-time for the first pass. Due to the adaptation in the second pass, a faster setup, e. g. tighter beams, can be used for the second pass. Therefore the second pass only needs 2.5 times real-time. In the first pass 58% of the Gaussians were evaluated per frame whereas in the second pass it were only 43%. The word error rate (WER) of the first pass is 35.7% and 29.4% for the second.

3.3 Speaker Independent Decoding

In this section I describe the experiments with the three investigated Gaussian selection methods, which I performed with the first pass of the decoder, and show the results.

3.3.1 Projection Search

For the investigation of the Gaussian selection using projection search (see section 2.4), the first pass of the decoder was executed with hyperplane distances of 1.5, 2.0, 2.5, 3.0 and 3.5 bounding 3, 6, 9 and 12 coordinate axes. It should be sufficient to bound only a few coordinate axes, because LDA was used on the feature space beforehand and sorted the dimensions according to their discriminative power, even though the STC may undo some of the sorting. The bounding is processed independently for every Gaussian mixture, when the state has to be evaluated. The data structure used for the selection of the Gaussians is simply a plain list that contains the indices of the Gaussians selected in the current Gaussian mixture. Since this data structure is produced while running the decoder, no disk space and only a few KB of main memory are needed. If no Gaussians are left after the bounding of a coordinate axes, a back-off vector is used to approximate the value of the Gaussian mixture. The Gaussian used as a back-off is specified by selecting the Gaussian that is nearest to the feature vector according to the l_1 norm in the currently selecting dimension out of the list of Gaussians that was selected by bounding the previous dimension.

As can be seen in figure 3.3(a) the performance of the systems get worse for smaller thresholds and larger numbers of bounded coordinate axis. That is reasonable, since with a smaller threshold and a larger number of bounded coordinate axes, less Gaussians are used to calculate the score. On one hand the performance decreases quiet fast, if the threshold gets beneath a certain threshold or above a given number of bounded axis. On the other hand the performance of the decoder is almost independent of the dimensions that are bounded, when the threshold is large enough. For a threshold of 2.5 or greater the systems in the experiments with projection search reach the performance of the baseline system nearly independent of the number of bounded coordinate axes (see figure 3.3(a)). Figure 3.3(b) It shows that if this boundary threshold is used (here: 2.5) the speed can be improved.

Figure 3.3(e) shows, that the bounding of more dimensions results in Gaussians with a better quality. Therefore less Gaussians have to be used to achieve the same word error rate, what results in less computation time for the evaluation of this Gaussians. This is probably due to the LDA, since many Gaussians are sorted out in the lower dimensions due to the sorting of the dimension according to their discriminative power. Therefore using larger threshold results in a better approximation of the nearest Gaussians. The speed up can be improved further by using a smaller threshold with a small number of bounded coordinate axis. With

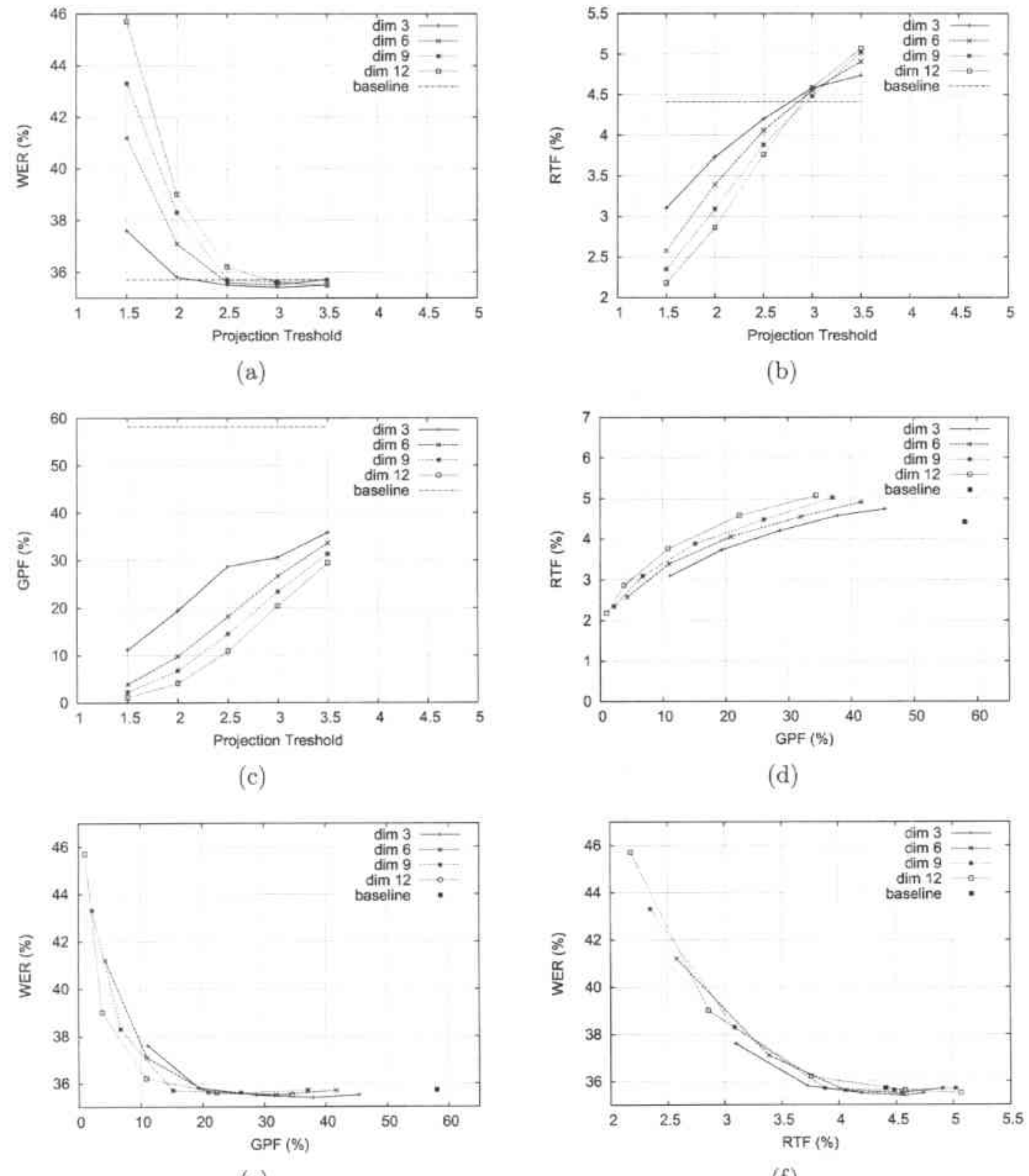


Figure 3.3: Results of first pass with projection search

this result the statement of Ortmanns [Ort98, OFN97], that it is sufficient to bound only a few coordinate axes, can be verified. But it has to be kept in mind that this is probably the case, because LDA was performed beforehand. It is important, that the used threshold is selected that way, that the performance is close to the performance of the baseline system, since otherwise the system quickly becomes slower as the baseline system. This is due to the additional computation time needed for bounding of the coordinate axis. For big thresholds the real-time factor is even worse if more coordinate axis are bounded, what also is caused by the overhead for bounding the coordinate axis since the number of Gaussians increases linearly (see figure 3.3(c)). The effect of the overhead can also be seen in the dependency between the real-time factor and the percentage of evaluated Gaussians per Frame. The real-time factor and the percentage of evaluated Gaussians are not proportional (see figure 3.3(d)). This is caused by the fact, that the overhead becomes less with a smaller threshold, since less Gaussians are within the bounds of the bounded dimensions. Therefore less Gaussians have to be considered in the higher dimensions and less dimensions have to be bounded before a back-off vector is used, what results in less overhead but also worse performance.

Figure 3.3(f) shows, that the best trade-off between the real-time factor and the word error rate can be achieved with a small number of bounded coordinate axes. The maximum speed up for the decoder with projection search without significant loss of accuracy is 16% while 19% of the Gaussians were evaluated by the scoring function.

3.3.2 BBI

To evaluate the performance, time and memory behavior of the big-BBI algorithm, as described in section 2.2, in the speaker independent first pass of the decoder BBI trees with the depths $6, 8, 10$ and 12 and relative thresholds $0.2, 0.3, \ldots, 0.7$ were created over all codebooks of the speech recognizer. This BBI trees were used to select the most significant Gaussians.

Using a BBI tree for the Gaussian selection in the first pass of the speech recognition only $3-20\%$ of the Gaussians are used by the scoring function. Figure 3.4(a) shows that with a increasing threshold and a increasing depth of the BBI tree the percentage of Gaussians per frame decreases. That is what was expected, because in a deeper tree a single leaf contains less Gaussians and only the Gaussians of one leaf are used by the scoring function. If the threshold is larger a smaller part of the Gaussians is used to define in which leafs it is located, so with a increasing threshold the number of leafs a Gaussian is located in decreases and therefore the average number of Gaussian per leaf decreases, too.

Although the number of Gaussians in the first pass can be reduced to $3-20\%$, figure 3.4(b) shows that the real time factor only decreases to $65-85\%$ of the baseline system. The conclusion is that the evaluation of the Gaussians is not the only part of the speech recognizer that needs a lot of computation time, but as figure 3.4(c) shows, the reduction of the real time factor for a certain depth of the BBI trees is nearly proportional to the reduction of the evaluated Gaussians per frame for thresholds that are not extreme concerning the BBI algorithm. If it would be possible to reduce the Gaussians per frame to very few, the real time factor would still be about 2.8. With a larger threshold or a greater depth of the BBI tree the real-time factor decreases in almost the same manner as the number of Gaussians. Unexpectedly the gradient of the real-time factor for decoders with BBI trees of depth 12 gets very low for thresholds of 0.4 and higher (see figure 3.4(b)).

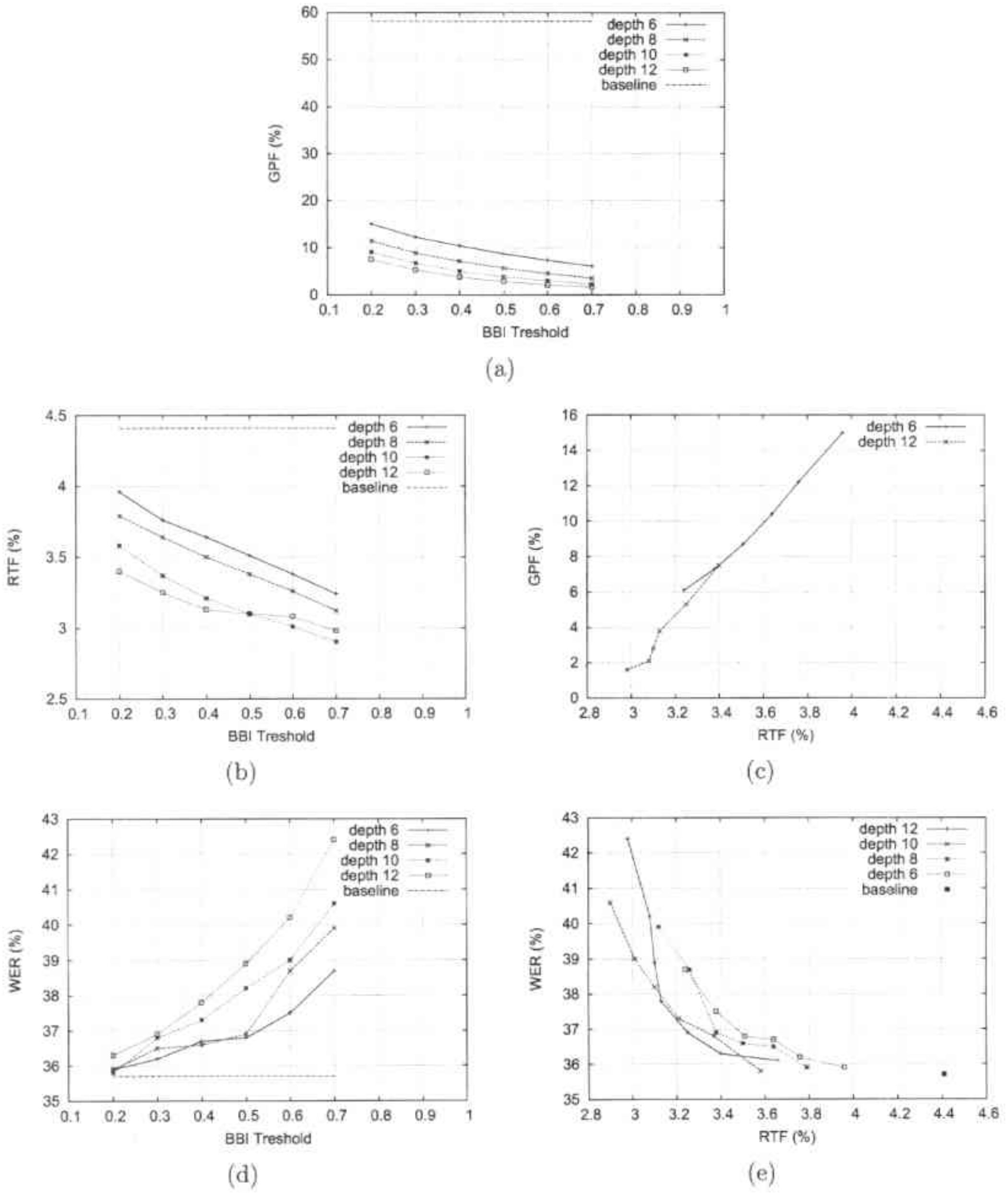
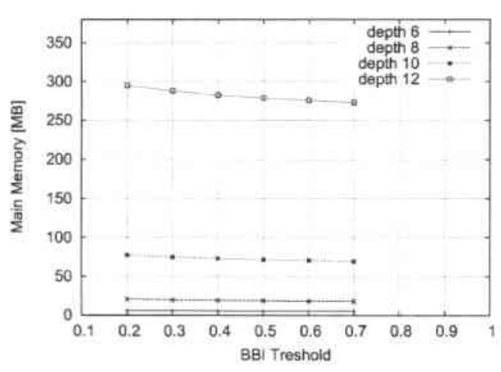
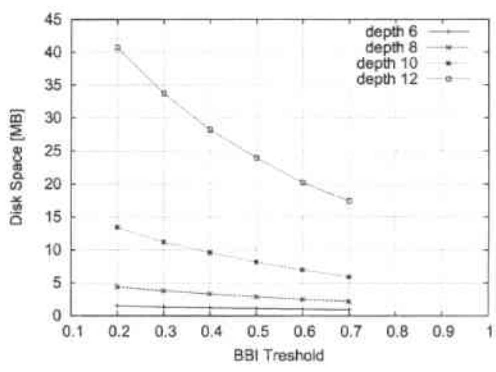


Figure 3.4: Results of first pass with BBI trees

Since for deeper BBI trees and larger thresholds more codebooks exist that have no Gaussians, which are located in a certain leaf, the back-off strategy is used more



(a)



(b)

Figure 3.5: Memory for BBI

often. The results for BBI trees with depth 12 suggest that the Gaussians chosen by the back-off strategy do not result in a good probability and therefore the speech recognizer has to track more paths, what results in a slower decoding. In contrast to that it seems that in general the deeper trees have a better trade-off between word error rate and speed up.

During the examination of the percentage of Gaussians per frame, it was suspected that the word error rate of the first pass is worse for BBI trees with larger thresholds, i. e. smaller boxes. As can be seen in figure 3.4(d) this is absolutely the case. It can also be seen that the word error rate increases with the depth of the BBI trees. In addition to that the gap of the word error rate between the BBI trees of different depths becomes bigger for larger thresholds. Whereas the maximum degradation of the word error rate is only about 0.5% for a threshold of 0.2 the degradation increases to 7% for a threshold of 0.7. Figure 3.4(e) shows, that the speed up of the system can be higher with the same word error rate for deeper BBI trees. Therefore a larger depth of the BBI tree should be preferred to a larger threshold, when making the decoder faster. Since with the depth of the BBI trees the amount of main memory and disk space that is needed to store the BBI trees in addition to the main memory and disk space of the baseline system increases very fast (see figures 3.5(a) and 3.5(b)), deep BBI trees can only be used, if enough main memory and disk space is available. The needed amount of memory does also increase with the value of the threshold, but not as significant as with the depth. A system with a BBI tree of depth 12 needs up to 300 MB of additional main memory and between 15 and 40 MB of additional disk space, whereas a system with a BBI tree of depth 6 only needs about 5 MB of additional main memory and about 1 MB additional disk space. If the system is restricted on the amount of memory, only flat BBI trees can be used and therefore the speed up for the same word error rate is smaller. The maximum speed up without a significant loss of accuracy is 18% for a BBI tree with depth 10 and a threshold of 0.2.

3.3.3 Clustering (Vector Quantization)

Initial Parameters

When using the clustering implementation of this thesis for Gaussian selection during a decoder run of a speech recognizer, four different parameters have to be optimized. This parameters are the number of clusters, the distance measure used during the clustering, the number of evaluated clusters during decoding and the back-off strategy. Since divergence measures the similarity of two Gaussians, it seems to be the most promising distance measure. Therefore divergence was used as distance measure during clustering in the first experiments. It can be calculated as

$$d(\mu_{\omega_1}, \mu_{\omega_2}) = \sum_{j=1}^{K} \left(\frac{\sigma_{\omega_1 j}^2 + (\mu_{\omega_1 j} - \mu_{\omega_2 j})^2}{\sigma_{\omega_2 j}^2} + \frac{\sigma_{\omega_2 j}^2 + (\mu_{\omega_1 j} - \mu_{\omega_2 j})^2}{\sigma_{\omega_1 j}^2} \right) \tag{3.1}$$

Besides the distance measure a initial set of cluster centroids is needed for the k-means like algorithm (see attachment A.3). To find the best initialization for the cluster centroids a view experiments have been made. The best clustering results have been achieved, when using random Gaussians out of the Gaussian mixtures as initial centroids.

Back-off Strategy

Every time the clusters, that have to be evaluated during decoding, do not contain any Gaussian of a Gaussian mixture, that has to be evaluated, a back-off strategy has to be used, to specify a value for this Gaussian mixture. Using the initial setting described above, we tried to find the best back-off strategy. Three different back-off strategies have been investigated. One simple possibility is to use a fixed value as back-off, what is easy to realize. A second approach is to use the value of the $N + 1$ cluster centroid at the coordinates of the feature vector, when evaluating the top N clusters. The third approach, that was evaluated, is to use a single Gaussian of the Gaussian mixture to calculate a back-off value. Therefore a special Gaussian of the Gaussian mixture is used, which is nearest to the cluster centroid. This assignment of the nearest Gaussians to the cluster centroids can be calculated during the preprocessing.

For the evaluation of the back-off strategies 1024 clusters were used since 1024 leafs performed best with the BBI. As a constant back-off 70 was used, which was found empirically. To see the influence of the back-off strategies on the performance of the first pass of the decoder, the decoding was processed using the best 16, 32, 64, 128 and 256 clusters. Figure 3.6(a) shows, that the choice of a good back-off strategy is really important, since the decoder gets worse, when evaluating less clusters and therefore using the back-off strategy more often. For 64 evaluated clusters the average back-off value of, when using the $N + 1$ centroid as back-off, is the same as the constant back-off. Therefore it can be seen, that using a fixed value is worse than using a variable value. In addition to that the average value of the variable back-offs changes with the number of evaluated Gaussians per frame (see figure 3.6(b)), which is dependent on the percentage of clusters, that have to be evaluated during the decoding. Therefore for each ratio another back-off value would have to be used. Using the value of the $N + 1$ cluster as a back-off value is much better than a constant value, but it is still much worse than using the value of the Gaussian nearest to the cluster centroid as a back-off. With this strategy it is possible to evaluate only 2.6% of the clusters without any degradation of the word error rate.

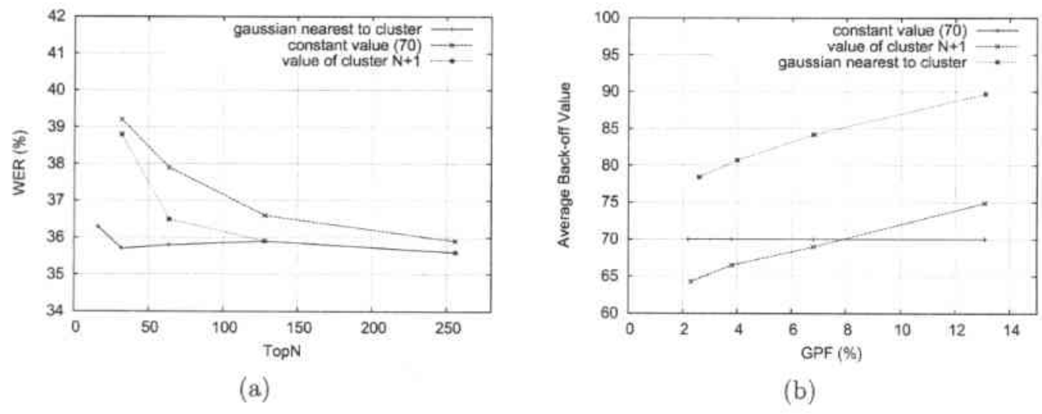


Figure 3.6: Back-off strategy for clustering

Clustering with Divergence as Distance Metric

Using the nearest Gaussian as the optimal back-off strategy, it is still necessary to know the appropriate number of clusters and the optimal number of clusters, that should be evaluated during decoding. For this reason another test was made using the nearest Gaussian as back-off. This time the nearest 8, 16, 32, 64 or 128 clusters out of 512, 1024 or 2048 clusters were evaluated during decoding.

To store the clusters the cluster set data structure, described in attachment B.2, is used. The amount of memory, that is needed for it can be seen in table 3.1. The amount of main memory and disk space increases with a larger number of clusters, but it is almost independent of the used distance measure and the acoustic models. Compared to the memory needed for the overall system, especially the language model, the amount of memory needed for the clusters is relatively small and should not be a restriction on the useability of clustering as Gaussians selection.

Figure 3.7(a) shows, that all decoders get worse if the number of clusters, evaluated during the decoding, gets smaller. Surprisingly the performance is mostly dependent on the number of evaluated clusters but not on the total number of clusters. When evaluating the same number of clusters for different total numbers of clusters, the

Number of Clusters	Main memory	Disk space
512	2MB	10MB
1024	5MB	20MB
2048	9MB	40MB

Table 3.1: Main memory and disk space needed for the cluster set data structures

percentage of evaluated Gaussians decreases. Therefore it seems reasonable that the decoder gets worse for a larger number of clusters, when evaluating the same number of clusters, but this might be compensated by the better granularity and the good back-off strategy. With a better granularity the Gaussians that are selected according to the nearest clusters give a better value (see figure 3.7(b)).

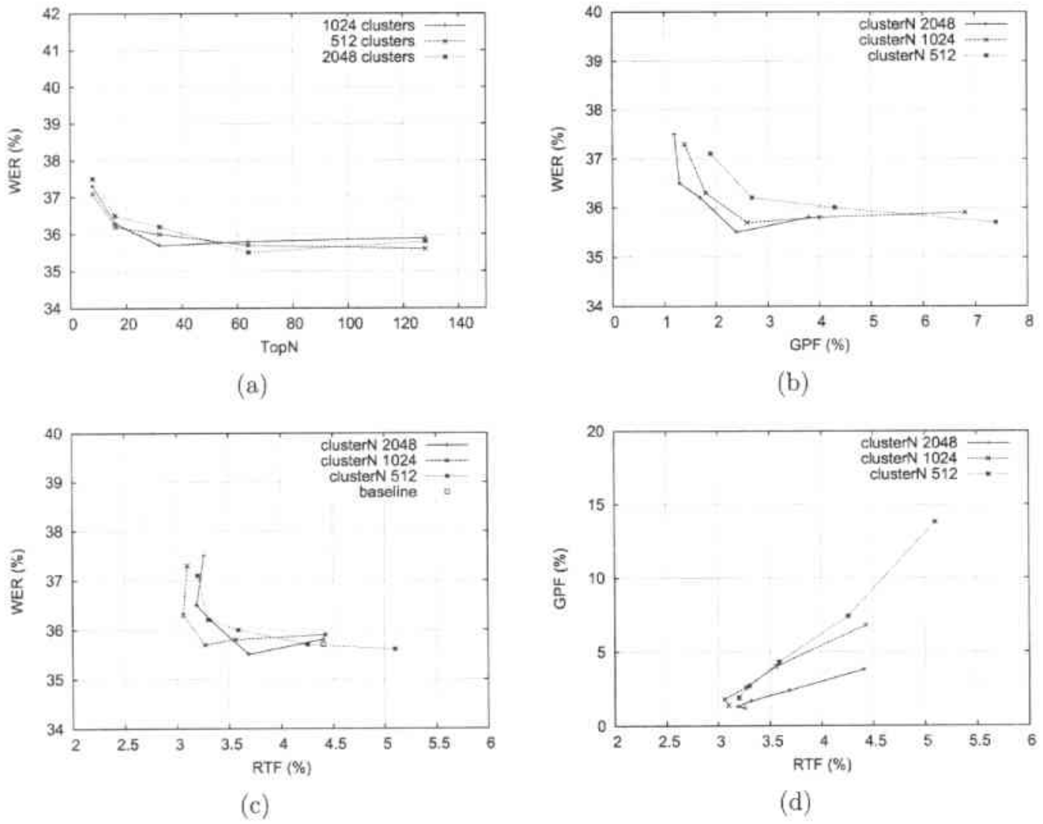


Figure 3.7: Results of first pass with divergence clustering

Although it is sufficient, to use less Gaussians when using more clusters, the system does not get faster, when using too much cluster. This is due to the fact, that for searching the nearest N clusters, all cluster Gaussians, which represent the cluster centroids have to be evaluated for each frame. This overhead slows down the system, when the number of clusters gets close to the number of Gaussians evaluated per frame. Therefore for the used system it is best to evaluate 32 out of 1024 clusters (see figure 3.7(c)). It can be seen, that with a certain minimum number of evaluated Gaussians, the system gets quickly worse but the real time factor does not decrease any more. This is due to the fact, that a system which evaluates only few clusters evaluates few Gaussians and often uses the back-off strategy. Since this values are worse for finding the best path, more paths have to be followed during the decoding. Therefore the system does not get faster although it evaluates less Gaussians. In figure 3.7(d) can be seen, that the number of evaluated Gaussians is nearly proportional to the real-time factor for a certain number of clusters as long as the number of Gaussians does not decreas below a certain level. This might be due to the fact, that the approximated probabilities for the Gaussian mixtures get to worse and therefore more paths have to be followed, what results in a higher computation time. If the number of evaluated Gaussians could be reduced to very few, a system with 1024 clusters would still need about 2.5 times real-time, whereas the best system without a significant loss of accuracy evaluates 2.6% of the Gaussians and gains a speed up of about 25%. This system evaluates 32 out of 1024 clusters.

Clustering with Euclidean Distance as Distance Metric

Clustering with divergence as distance metric showed, that a lot of overhead is produced, when using more clusters although the quality of the selected Gaussians increases. Therefore it might be better, to use Euclidean distance as distance measure during the search for the nearest N clusters. This would lead to less overhead, since the calculation of the Euclidean distance is faster than calculating the divergence. When using Euclidean distance for the search of the nearest N clusters, it seems reasonable to use Euclidean distance for the clustering too. Due to the simpler distance measure during the clustering the quality of the selected Gaussians might get worse. Therefore it has to be experienced if the reduction of overhead is larger than the additional computation time needed due to the worse quality of the Gaussians. To investigate this relationship tests with 1024 and 2048 clusters were performed using Euclidean distance. During the decoding 8, 16, 32 and 64 clusters were evaluated, while the nearest Gaussian was used as back-off.

In contrast to clustering with divergence the WER does not only depend on the number of evaluated clusters when using Euclidean distance. The WER depends on the ratio between evaluated clusters and the number of the overall clusters. For the same ratio the same performance of the system can be achieved (see figure 3.8(a)). When using more clusters the quality of the clusters does not increase and therefore more clusters have to be evaluated to get the same performance. This might be due to the bad quality of the clusters caused by the simpler distance measure. When evaluating the same ratio of clusters the number of evaluated Gaussians is the same and they results in the same performance (see figure 3.8(b)). That means, that the quality of the resulting Gaussians is about the same and the time needed for the evaluation of the Gaussians is also about the same. Figure 3.8(c) shows that the result of this is, that the system, which has a smaller number of clusters, is faster without loss in word error rate, since less computation time is needed to find the nearest clusters. When using a system with more clusters, the evaluated back-offs are closer to the observation vector than for a system with less clusters. This results in a larger speed up for the system before it slows down again due to the worse path search. When evaluating 32 out of 1024 clusters a speed up of 32% can be achieved with 2.6% evaluated Gaussians.

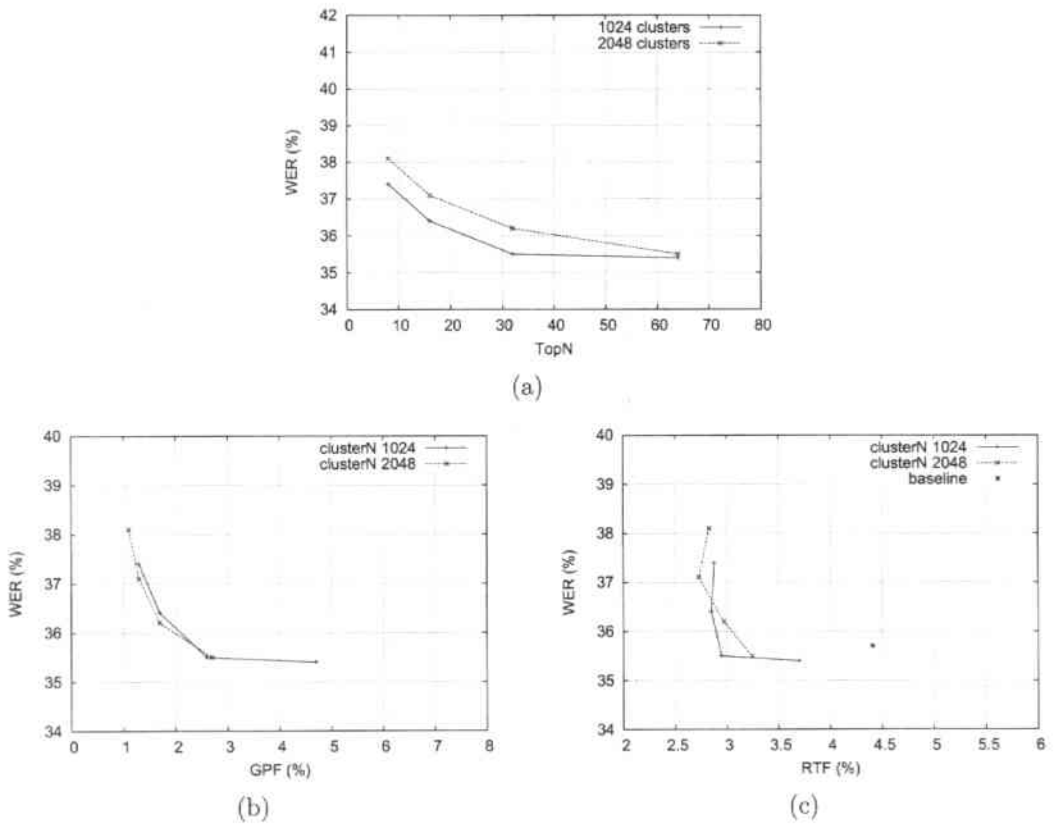


Figure 3.8: Results of first pass with Euclidean distance clustering

3.4 Speaker Adaptive Decoding

In this section I first investigate the influence of the speaker independent hypothesis on the second passs. Then I describe the experiments with the three investigated Gaussian selection methods, which I performed with the second pass of the decoder, and show the results.

3.4.1 Influences of the Speaker Independent Hypothesis on the Second Pass

The hypothesis of the first pass, which are used for the speaker adaptation in the second pass, might have an influence on the performance of the second pass. To experience the influences, the second pass was processed without Gaussian selection but using the hypothesis of the first pass with BBI for the speaker adaptation.

The Gaussians per frame used during the second pass are nearly the same for the baseline system and all systems with BBI hypothesis. For a smaller number of Gaussians in the first pass as a result of a larger threshold, the number of Gaussians in the second pass becomes only slightly larger (see figure 3.9(a)). This suggests that the speaker adaptive decoder has to compensate the worse results of the first pass by evaluating more paths.

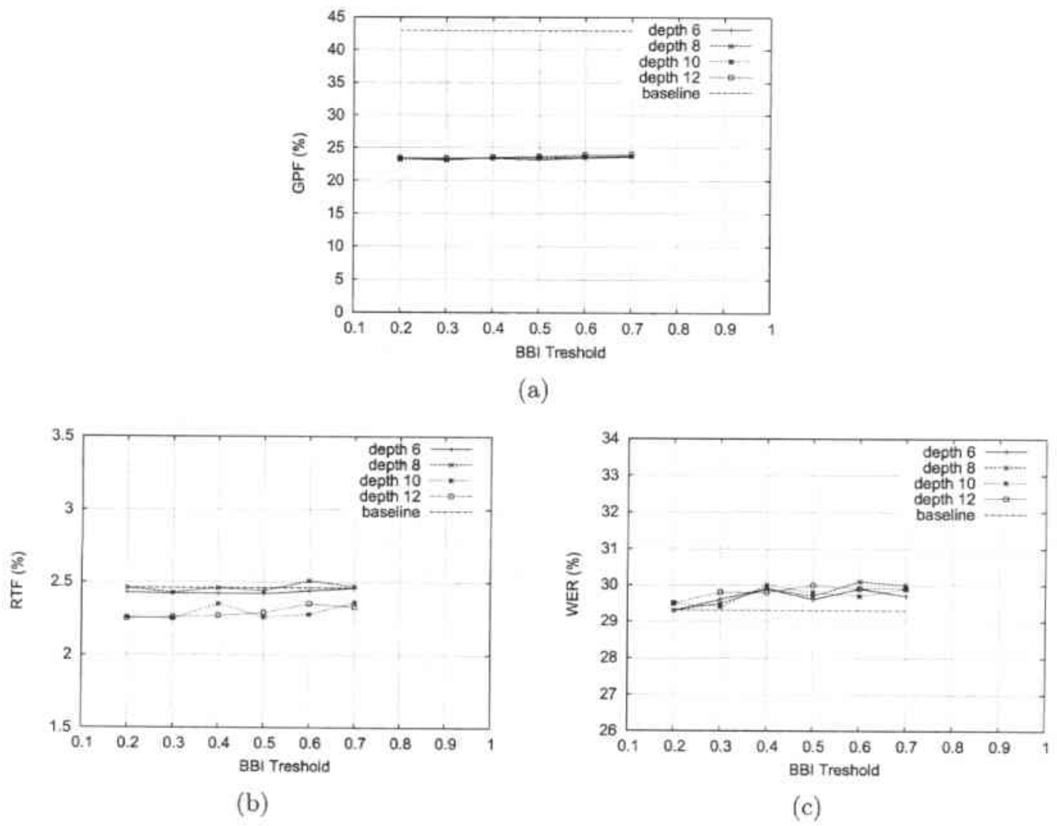


Figure 3.9: Results of second pass of baseline with hypothesis of first pass with BBI

The real-time factors for the second pass are nearly the same for all tests (see figure 3.9(b)). Probably the lines of the tests with different depth are not in the right order because of different computers on which the tests were processes. The decoder runs of the BBI tests in total are slightly faster than the baseline system, although they do not use BBI trees in this second run.

Although the degradation of the word error rate in the first pass is significant the word error rate after both passes decreases only 0.7 at most (see figure 3.9(c)). But that means that the hypothesis that are produced by the speaker independent decoder and used for the speaker adaptation in the second pass influence the behavior of the speaker independent pass. Therefore the behavior of the different Gaussian selection methods on the speaker adaptive pass could be better compared, if the same hypothesis are used for all speaker adaptations in the tests of the second pass. For tests with thresholds of 0.2 and 0.3 there is nearly no degradation, but the maximal speed up is already 8%. The average degradation of the word error rate for thresholds of 0.4 and higher is only 0.5. Therefore the system still can be made a bit faster by using higher thresholds with only little loss in accuracy. The results for tests with a threshold of 0.4 are not only influenced by the Gaussian selection. They are worse than expected because of a problem during speaker adaptation. For this systems some of the Viterbi paths during speaker adaptation could not be traced properly because of slightly to tight beams and therefore the speaker adaptation is a little bit worse, what results in a worse word error rate. The different speaker adaptations and there influence on the second pass can also be avoided by using the same hypothesis for all decoder runs of the speaker adaptive pass. Therefore for the following test of the speaker adaptive pass the speaker independent hypothesis of the baseline systems first pass will be used.

3.4.2 Projection Search

For the evaluation of the projection search in the speaker adaptive pass the same implementation and parameters were used as in the first pass (hyperplane distances 1.5, 2.0, 2.5, 3.0, 3.5 bounding 3, 6, 9 and 12 coordinate axes). Since the projection search is working on the feature space without any precalculations, it can simply be applied on the adapted feature space without adaptation. Therefore projection search is easy to use for the second pass. For the adaptation of the feature space the hypothesis of the baseline systems first pass were used to be able to compare the results with other methods in the second pass. To speed up the speaker adaptation projection search was also used during the alignment for the speaker adaptation.

The systems with projection search have the same characteristics in the second pass as in the first. Again with smaller hyperplane distances and more bounded dimensions the performance of the system gets worse (see figure 3.10(a)). Like in the first pass, there is a minimum threshold of about 2.5 and a maximum number of bounded coordinate axes onto which the performance stays constant. Above this thresholds the performance decreases very quickly. Using the minimum threshold of 2.5 no speed up of the system can be achieved (see figure 3.10(b)).

The system might have a slightly better performance, if no projection search would have been used during the speaker adaptation and therefore a speed up could be

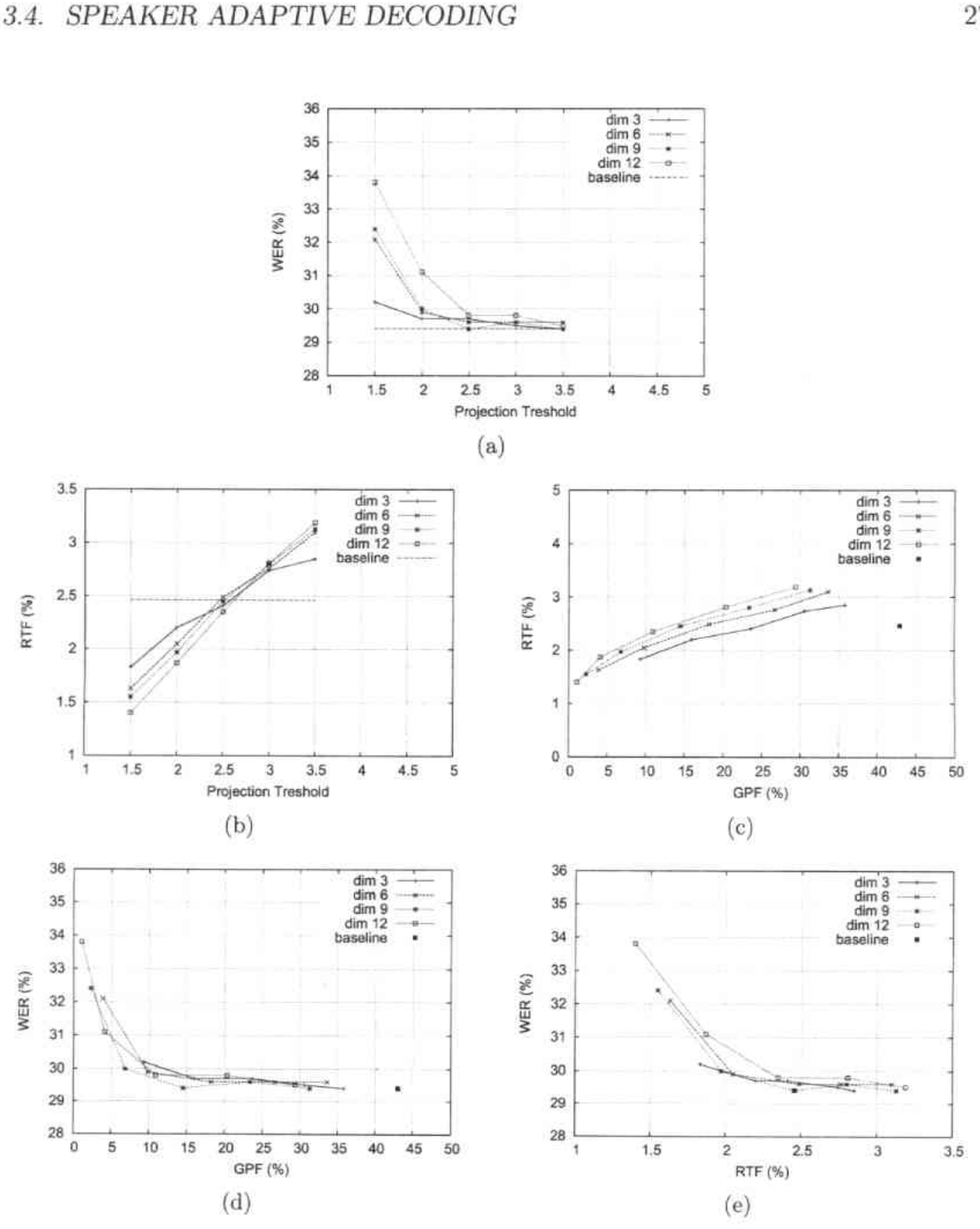


Figure 3.10: Results of second pass with projection search

achieved. This has to be investigated in more detail in future experiments. Since the projection search is based on the fact, that the dimensions of the feature space are sorted according to their importance, the slower systems could also be a result of the disturbed sorting of the dimensions after speaker adaptation. This could cause, that the overhead of the projection search is higher, since more dimensions have to be bounded before the number of the most significant Gaussians reduces significantly. The disturbed sorting of the dimensions also explains the high overhead for all parameters and therefore the proportionality between the percentage of evaluated Gaussians and the real-time factor (see figure 3.10(c)), which is unlike the first pass. For the quality of the Gaussians, it does not matter how many dimension are bounded. (see figure 3.10(d)). This is probably due to the disturbed sorting of the dimensions caused by the speaker adaptation (FMLLR, MLLR, SAT-trained models). Since less Gaussians are outside the bouding hyperplanes in the lower dimensions, the overhead for bounding does not reduce, when using a small threshold. When using a smaller threshold and a appropriate small number of bounded coordinate axes, the speed of the system can be improved. Figure 3.10(e) shows, that the maximum speed up with a little but not significant loss of performance is 16% while 14% of the Gaussians were evaluated.

3.4.3 BBI

Since the acoustic models of the second pass are modified during the speaker adaptation, BBI trees with this acoustic models should only be used during the alignment for the speaker adaptation. Since during the speaker adaptation the scoring function is called for every Gaussian mixture separately instead of for a set of Gaussian mixtures as during the decoding, the overhead of evaluating the BBI trees would slow down the system. Therefore it makes no sense to use the BBI trees during the speaker adaptation. To use the BBI trees for the decoding of the second pass, speaker-adapted BBI trees should be used. The first method to achieve speaker adapted BBI trees would be, to adapt precalculated BBI trees to every speaker after the speaker adaptation. Since no algorithm is known, that performs this adaptation, it is not possible. The second method would be to build new BBI trees after each speaker adaptation during the decoding. The second speaker adaptive pass distinguishes between 36 speakers, for which an adaptation is performed. Building a small BBI tree with depth 6 and a threshold of 0.7 already needs more than two minutes. Therefore it would take over 80 minutes to build BBI trees for all speakers. This is about 50% of the overall decoding time of the baseline system. Therefore a speed up of more than 50% for the calculation of the acoustic scores would be necessary to get a speed up for the overall system. That does not seem to be possible and therefore it would slow down the system instead of speeding it up. Nevertheless, it should be possible to get a speed up when building BBI trees on the adapted models if a large amount of audio data has to be processed for every speaker.

It is possible to use BBI trees build on the acoustic models of the second pass, but without adaptation. Therefore the second pass of the decoder was evaluated using BBI trees without adaptation. Due to the missing adaptation the BBI trees have to be somehow robust against the transformations processed during the speaker adaptation. This is achieved by using flatter BBI trees with smaller thresholds, since more Gaussians are in each leaf then and the transformations during the speaker adaptation have not such a big influence on the structure of the BBI trees. For tests BBI trees with depths 10, 8 and 6 and thresholds 0.1,...,0.4 were used.

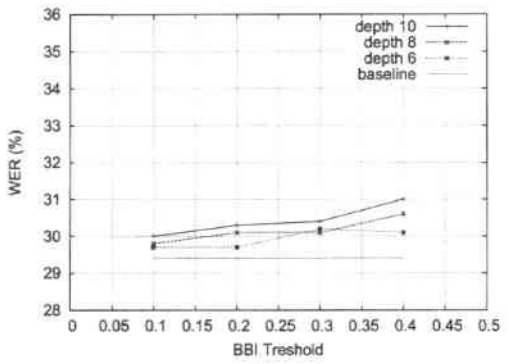


Figure 3.11: Results of second pass with BBI

Figure 3.11 shows, that the performance of the baseline system can be achieved with a BBI tree of depth 8 and a threshold of 0.1 or with a BBI tree of depth 6 and a threshold of 0.2. This verifies the assumption that flatter BBI trees with smaller thresholds have to be used in the second pass compared to the first pass. Using even smaller thresholds would not further speed up the system. When using a BBI tree with depth 8 and a threshold of 0.1 the decoder needs 5 MB of additional disk space and 22 MB of additional main memory during decoding, whereas the decoder only needs 1 MB additional disk space and 6 MB of additional main memory when using the BBI tree with depth 6 and a threshold of 0.2. When using one of these systems, about 13% of the Gaussians are evaluated, what is about the same amount as in the first pass. The speed up of these systems is 20%.

3.4.4 Clustering

Clustering with Divergence as Distance Metric

The clusters, which are precalculated on the acoustic models of the second pass, could be used during the alignment for the speaker adaptation. Due to the implementation of the speaker adaptation, the scoring function is processed for every Gaussian mixture separately and therefore the nearest N clusters have to be found for every Gaussian mixture. Since this results in an overhead instead of an speed up, the clusters should not be used during the speaker adaptation.

When using clustering for Gaussian selection during the decoding of the second pass, the clusters have to be updated for every speaker. This means, that the Centroids, the assignment of the Gaussians to the clusters and the assignment of the back-off vectors should be updated. Since the reassignment of the Gaussians and the back-off vectors is slow, only the cluster centroids can be updated without additional overhead. They are recalculated based on the assignment of the Gaussians to the centroids, which was calculated on the unadapted acoustic models, and the Gaussians of the adapted acoustic models. To see, if this results in a better performance for the system experiments with 1024 clusters build on the acoustic models of the second pass have been performed. They were used with and without adaptation for every speaker. For the tests 32 clusters were evaluated. The word error rate without adaptation is 29.7%, whereas adapting the centroids results in a word error rate of 29.2%. This shows, that the word error rate can be improved, when updating the centroids, without a lot of additional computation time.

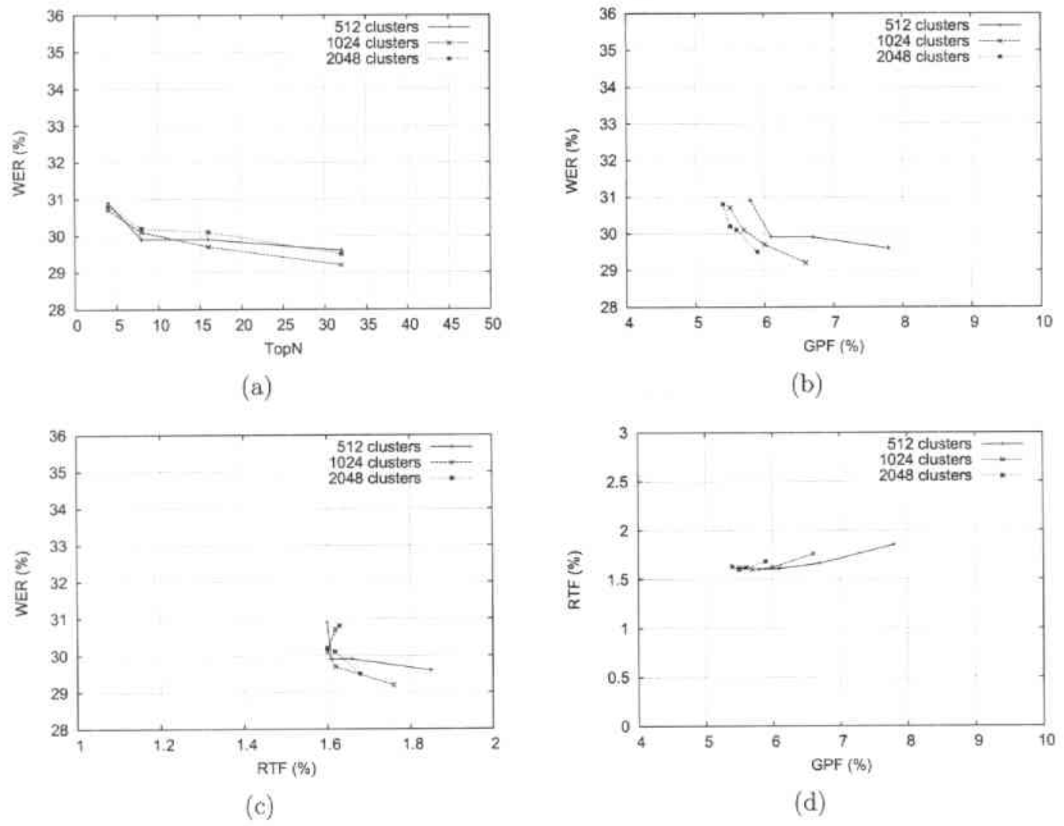
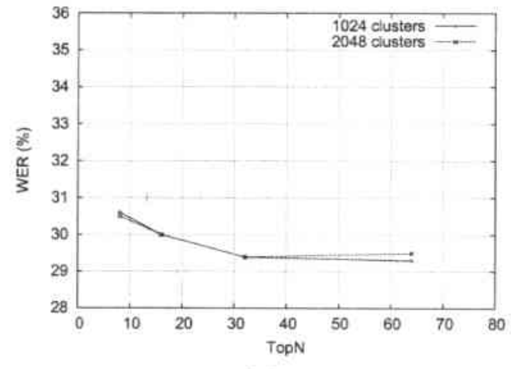


Figure 3.12: Results of second pass with divergence clusering

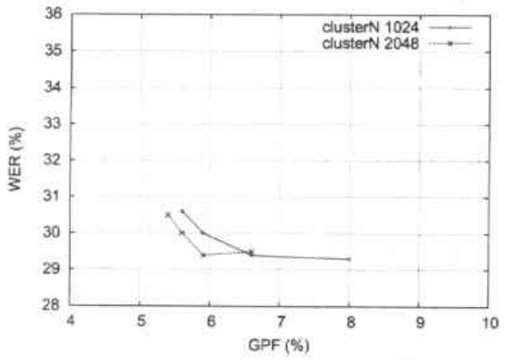
Using the centroid adaptation to adapt the clusters to every speaker, experiments were performed using 512, 1024 and 2048 clusters. During the experiments 4, 8, 16 and 32 clusters have been evaluated. Figure 3.12(a) suggests that the word error rate of the system mainly depends on the number of evaluated cluster, but not on the overall number of clusters in the cluster set. This could be caused by the fact, that reducing the number of Gaussians and increasing the quality of the Gaussians compensates each other, when using different numbers of clusters. Looking at figure 3.12(b) it can be seen, that using more clusters results in Gaussians with a better quality and therefore less Gaussians can be used to get the same performance, what results in less computation time for the evaluation of the Gaussians. Since it needs more computation time to find the nearest clusters, when using more clusters the speed up of getting better Gaussians does not result in a speed up for the overall system (see figure 3.12(c)). At some point, the systems get a worse performance without further speed up, what probably is due to the fact, that a worse system needs to consider more paths and therefore gets slower, although evaluating less Gaussians. This can also be seen in figure 3.12(d) . This might be due to the fact, that more paths are considered during the decoding. The system with the best speed up without loss in word error rate achieves a speed up of 36%, when evaluating 16 out of 1024 clusters.

Clustering with Euclidean Distance as Distance Metric

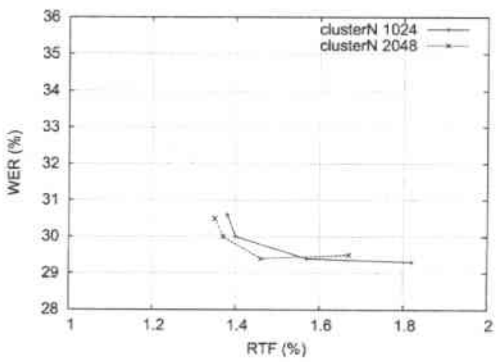
The overhead for searching the nearest N clusters can be reduced, when using Euclidean distance instead of Mahalanobis distance.



(a)



(b)



(c)

Figure 3.13: Results of second pass with Euclidean distance clustering

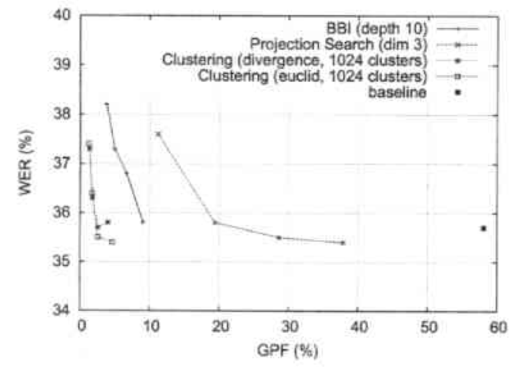
To investigate the reduction of the overhead for searching the nearest clusters with Euclidean distance and the resulting quality of the selected Gaussians, experiments with 1024 and 2048 clusters have been performed. For the clustering Euclidean distance was used too. During the experiments 8, 16, 32 and 64 clusters have been evaluated. For the adaptation of the clusters to the different speakers the centroids are updated after each speaker adaptation.

During the experiments it could be seen, that the performance again only depends on the number of evaluated clusters (see figure 3.13(a)). This is different to the first pass, where the performance depends on the proportion of the evaluated clusters. This might be due to the fact, that the update of the centroids results works better, when using more clusters. Since the same number of clusters can be evaluated to achieve the same word error rate for different numbers of overall clusters, less Gaussians have to be evaluated when using more clusters (see figure 3.13(b)). This means, that the Gaussians have a better quality, when using more clusters. Since less Gaussians have to be evaluated, the computation time needed for the evaluation of this Gaussians is also less. Figure 3.13(c) shows that this leads to a better speed up of the system, when using more clusters, although the overhead for finding the nearest clusters is higher then. The speed up that can be achieved is 40%, when evaluating 32 out of 2048 clusters with 5.9% evaluated Gaussians.

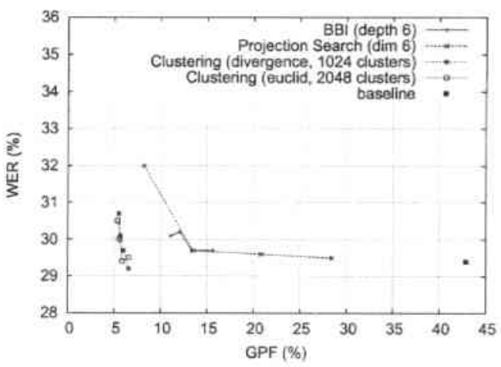
3.5 Comparison

When comparing the different Gaussians selection methods and trying to achieve a high speed up without increase in word error rate two things are most important. The first is the quality of the Gaussians, which are selected by the methods and the second is the computation time, that is needed to find these Gaussians. In figures 3.14(a) and 3.14(b) can be seen that in both passes using clusters for the Gaussian selection results in the best Gaussians, since least Gaussians can be used without loss in word error rate. In out setup it makes no difference, which distance measure is used for the clustering. During the first pass the BBI results in better Gaussians than the projection search, what was expected due to the more complex selection of the Gaussians. As can be seen BBI and projection search select Gaussians with the same Quality in the second pass, what might be due to the missing speaker adaptation of the BBI trees. With all methods it is possible to use at most one third of the Gaussians that are used by the baseline system.

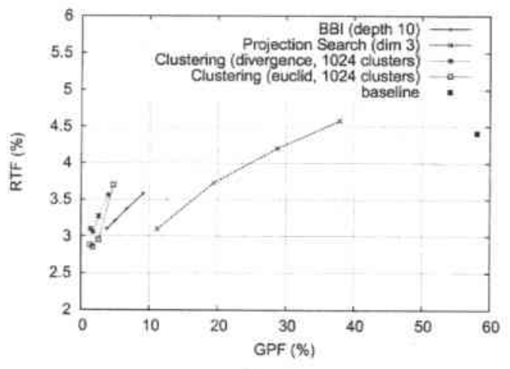
The speed up that can be achieved in evaluating less Gaussians has to be seen in relation to the overhead that is produced to select these Gaussians. To select less Gaussians, that give a good approximation for the value of the Gaussian mixtures more time has to be spend on the search for these Gaussians. As can be seen in figures 3.14(c) and 3.14(d) clustering is the slowest system, when evaluating the same number of Gaussians, what means that clustering has the highest overhead for finding the nearest Gaussians. The overhead depends on the distance measure as can be seen in the figures for both passes, since for the same number of Gaussians they have a quite large difference in the real-time factor. This difference gets smaller in the second pass. In the first pass, the system with the smallest overhead to find the same number of Gaussians is the projection search, whereas the system with the second most overhead is the BBI. Since better results can be achieved in the second pass, when using projection search with more bounded dimensions and using shallower BBI trees, the projection search has a slightly higher overhead as BBI.



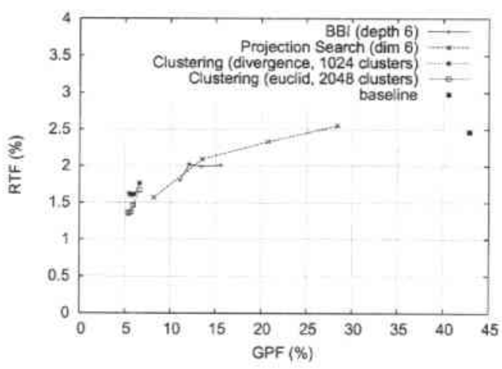
(a)



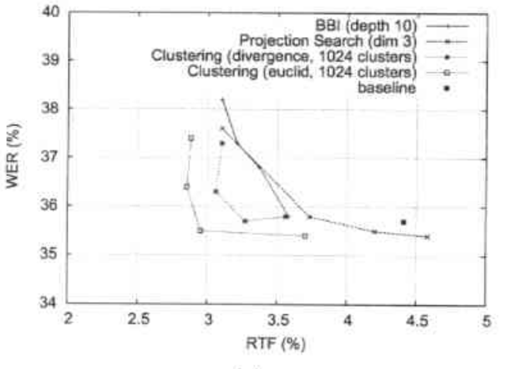
(b)



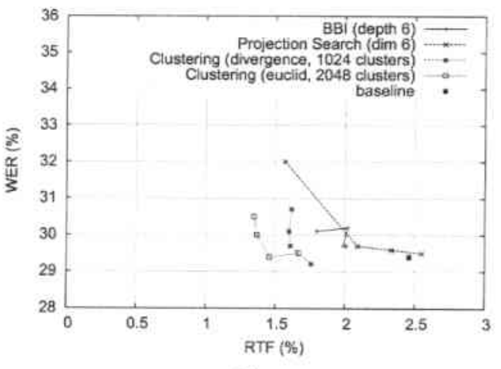
(c)



(d)



(e)



(f)

Figure 3.14: Best systems for both passes (left: first pass, right: second pass)

Since the Gaussians selected by the projection search or the BBI have the same Quality in the second pass, this results in a better speed up when using the BBI instead of the projection search (see figure 3.14(f)). Due to the fact, that the quality of the Gaussians, selected by the clustering, is much higher, a lot less Gaussians can be used to achieve the same word error rate. This means, that although the overhead is higher for the Gaussian selection with clustering, the speed up that can be achieved is higher for clustering than for BBI or projection search in the second pass. Since clustering with the different distance measures results in a similar quality of Gaussians and the overhead is a lot less when searching the nearest clusters with Euclidean distance, the highest speed up can be gained, when evaluating the clusters with Euclidean distance. The same behavior between the different Gaussian selection methods can be seen in the first pass (see figure 3.14(e)). Table 3.2 shows the speed up, that can be achieved for the different Gaussian selection methods, without loss in word error rate. When trying to build a system, that is faster than the baseline system and that can be worse than the baseline, the faster Gaussian selection methods might be better. When speeding up the system with a loss in word error rate, the overhead for the different methods stays about the same, but the number of selected Gaussians can be reduced more, when having more Gaussians before the reduction. This is the case for the faster methods and therefore the reduction in computation time needed for the evaluation of the selected Gaussians can be reduced more, what results in a better speed up than for the slower methods.

First pass				
	Proj. search	BBI	Clusters (diverg.)	Clusters (Eucl.)
WER	35.8%	35.8%	35.7%	35.5%
Speed up	16%	18%	25%	32%
GPF	19%	9.1%	2.6%	2.7%
Main memory	<<1MB	80MB	20MB	20MB
Disk space	0MB	13MB	5MB	5MB
Second pass				
	Proj. search	BBI	Clusters (diverg.)	Clusters (Eucl.)
WER	29.7%	29.8%	29.7%	29.4%
Speed up	16%	20%	36%	40%
GPF	14%	13%	6.0%	5.9%
Main memory	<<1MB	6MB	20MB	40MB
Disk space	0MB	2MB	5MB	9MB
Adaptable to spk.	-	NO	YES	YES

Table 3.2: Systems with the best speed up for the different Gaussian selection method

The differences of the methods between their behavior in the first and the second pass are partly caused by their ability of speaker adaptation. Since the clustering and the BBI are trained on the SAT-acoustic models, they should be updated for every speaker. The update of the centroids only seems to be sufficient for the update of the clusters and therefore the clusters are adaptable to the speaker, whereas it is not possible to adapt BBI trees. This results in a worse behavior for the BBI in the second pass compared to the projection search, which just works on the acoustic models without precalculation and therefore needs no speaker adaptation.

BBI		Cluster	
Number of leafs	Main memory	Number of clusters	Main memory
64	5 MB		
256	19 MB	512	10 MB
1024	72 MB	1024	20 MB
4096	280 MB	2048	40 MB

Table 3.3: Main memory needed for BBI and clustering

	BBI	Clusters (divergence)	Clusters (Euclid)
Speed up 1st pass	14%	25%	32%
WER 1st pass	35.9%	35.7%	35.5%
Speed up 2nd pass	20%	36%	36%
WER 2nd pass	29.8%	29.7%	29.4%
Main memory	21MB	20MB	20MB
Disk space	4 MB	5MB	5MB
Leafs/Clusters	256	1024	1024

Table 3.4: Speed up for the different Gaussian selection methods using the same amount of memory

The different methods and their different adaptability need different amounts of memory. Since the projection search works directly on the acoustic models, it needs only a few KB of main memory and no disk space, while the precalculated data structures of the other methods need at least some MB of disk space and main memory. The memory needed for clustering is about the same for the different distance measures and both passes. It mainly depends on the number of clusters (see table 3.3). The memory of a BBI tree also depends mainly on the number of leafs. When building a BBI tree with the same number of leafs than clusters, the BBI tree needs a lot more memory (see table 3.3). This is caused by the fact, that the Gaussians can be located in more than one leaf while the clusters are disjoint. When examining the memory, that is needed by the best systems for every Gaussian selection methods, no real conclusion can be drawn on the memory behavior and the resulting speed up of the system (see figure 3.2). Speed ups close to the best speed up can also be achieved, when using more or less leafs or clusters, what would result in very different memory amounts. Therefore it is always a trade-off between speed up and needed memory. As can be seen in table 3.4, higher speed ups can be achieved for the clustering when using the same amount of memory for the clustering and the BBI. That means, that the BBI needs more memory than the clustering. In addition to that the memory needed by the BBI increases a lot faster than the memory needed for the clusters, what again is due to the fact, that Gaussians can be located in multiple leafs. Besides that, the amount of memory that has to be used is more flexible, when using clusters, since any number of clusters can be used, while the BBI trees are balanced and therefore the number of leafs always is a power of 2.

Chapter 4

Conclusions and Future Work

During the evaluation of the different Gaussian selection methods, the quality of the selected Gaussians and the time needed for selecting the Gaussians were the main factors for the speed up of each system. The trade-off between this two factors varies for the different methods, what results in different speed ups. The speed up that can be achieved with a certain method is also dependent on the amount of memory, that is available Therefore it is also a trade-off between the speed up and the amount of used memory.

It is possible to build a system that needs only a few KB of additional memory for the Gaussian selection. This can be done in using projection search for the Gaussian selection. With projection search it is possible to speed up both passes of the speech recognizer without loss in word error rate. It does not even need to be adapted to the speakers, since it works directly on the acoustic models. A better speed up with projection search can be achieved, when sorting the dimensions according to their discriminative power, which is done beforehand by performing LDA. Less speed up can be achieved in the second pass, since the ordering of the dimensions gets worse due to the speaker adaptation and the speaker adaptive trained acoustic models in the second pass.

Using a system with a BBI tree for the Gaussian selection a higher speed up can be achieved than with the projection search. Since the BBI tree has to be precalculated some disk space and also some main memory is needed for this Gaussian selection. This BBI tree would have to be adapted to each speaker. Although this is not possible, using a not adapted BBI tree also a higher speed up can be achieved as with projection search in the second pass.

When using clustering with divergence as Gaussian selection an even higher speed up as with the BBI can be achieved while less memory is needed. During the evaluation of the clusters it is best to use the Gaussian of each Gaussian mixture that is nearest to a cluster centroid to calculate a back-off value. To use the clusters for the speaker adaptive pass it is sufficient to update the cluster centroids for each speaker. Searching for the nearest clusters during the decoding needs a lot of computation time, when using divergence as a distance measure, therefore it is better to use Euclidean distance for the clustering and the search of the nearest clusters.

Clustering with Euclidean distance gains the best trade-off between speed up and word error rate for large systems. With this method a total speed up for both passes of 34% can be achieved without loss in word error rate. This includes speaker adaptation in the second pass. It is possible to reduce the overhead for searching the nearest clusters further, what will result in a larger speed up. Besides the time, that is needed for the evaluation of the Gaussian mixtures, a lot of computation time is also needed to evaluate the complex language model, what bounds the speed up that can be achieved using Gaussian selection.

One possibility to reduce the overhead for finding the nearest clusters is to perform a second clustering with less clusters on top of the first clusters. During the decoding first the nearest clusters of the second layer are selected and according to this preselection the clusters of the first layer are selected. This would reduce the overhead, since less distances between each observation vector and the cluster centroids would have to be calculated, but due to the more complex selection of the nearest clusters of the first layer, more parameters have to be tuned to get a good speed up. It could also be performed with more than two layers building tree-structured clusters.

Different other possibilities should be tested on their ability to get more speed up. Instead of using Euclidean distance a weighted Euclidean distance could be used for the clustering or a split and merge algorithm could be used for the clustering instead of the k-means like algorithm. Both might result in better clusters without additional overhead during the decoding. It might also be possible to reduce the number of clusters and therefore the overhead for the selection in using intersecting clusters instead of disjoint clusters. Therefore more complex assignments of the Gaussians to the centroids should be tested. Intersecting clusters would probably need more memory. Also further test on the back-off strategy could be performed. It could be tested how high the loss in word error rate is, when using a single Gaussian of a Gaussian mixture as back-off instead of evaluating all Gaussians as back-off. Maybe a higher speed up can be achieved, when using more than one Gaussian as back-off.

During this thesis the first and second pass have been examined separately. It might be possible to get more speed up without loss in word error rate, when tuning both passes together. It does not seem necessary to have very good hypothesis after the first pass. Good hypothesis might be sufficient to perform a good speaker adaptation during the second pass. Therefore it should be tested if a better speed up can be achieved, when using a fast first pass and an accurate second pass. Instead of considering both passes together, it might also be useful to combine two Gaussian selection methods, to get a better speed up. For example a preselection with a small number of clusters or a shallow BBI tree and projection search based on this preselection could be performed. After tuning the parameters for the Gaussian selection it might be useful to revisit the other tuning parameters of the system.

It might be possible for example to widen the beams without significantly slowing down the system. This could even result in a better word error rate.

Besides speeding up the evaluation of the Gaussians during the decoding, it would also be possible to speed up the evaluation of the Gaussians during the alignment of the hypothesis for the speaker adaptation. Probably this has more negative effect on the word error rate than positive effect on the speed, but to be sure it should be investigated further. Another possibility to speed up the system even more is probably to speed up the evaluation of the language model, which needs a lot of the computation time during the decoding.

Appendix A

Mathematical Basics

A.1 Evaluation of Gaussian Mixtures

For a given observation vector x the probability for a state ω containing a multivariate Gaussian mixture with diagonal covariance matrices is defined as:

$$p(x|\omega) = \sum_{i=1}^{N_\omega} c_{\omega i} N(x, \mu_{\omega i}, \Sigma_{\omega i}) \tag{A.1}$$

The variable $c_{\omega i}$ is the mixture weight for the ith Gaussian of state ω. The weights of a state ω fulfill the equations $\sum_{i=1}^{N_\omega} c_{\omega i} = 1$ and $c_{\omega i} \geq 0$ whereas N_ω is the number of Gaussians of state ω. A single Gaussian $N(x, \mu_\omega, \sigma_\omega)$ can be calculated as:

$$N(x, \mu_\omega, \Sigma_\omega) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_\omega|}} \exp^{-\frac{1}{2}(x-\mu_\omega)^T \Sigma_\omega{}^{-1}(x-\mu_\omega)} \tag{A.2}$$

Therefore the log probability of a single Gaussian with a diagonal covariance matrix of dimension K can be written as

$$logN(x, \mu_\omega, \sigma_\omega) = -\frac{1}{2}[log((2\pi)^K \prod_{j=1}^{K} \sigma_{\omega j}^2) + \sum_{j=1}^{K} \frac{(x_j - \mu_{\omega j})^2}{\sigma_{\omega j}^2}] \tag{A.3}$$

A.2 Calculation of a cluster Gaussian

Given the observation vectors o_i, the mean vectors for K diagonal Gaussians belonging to a cluster can be calculated as

$$G_k : \mu_k = E(o_i)_{i=(k-1)N}^{kN-1} = \frac{1}{N} \sum_{i=(k-1)N}^{kN-1} o_i \tag{A.4}$$

whereas their covariances can be calculated as

$$\begin{aligned}\sigma_k^2 &= E\left((o_i-\mu_k)^2\right)_{i=(k-1)N}^{kN-1} = E(o_i^2)_{i=(k-1)N}^{kN-1} - E(\mu_k)^2 \\ &= \frac{1}{N}\sum_{i=(k-1)N}^{kN-1} o_i^2 - \mu_k^2 \end{aligned} \tag{A.5}$$

assuming that each Gaussian approximates the distribution of N observation vectors o_i. The mean vector and the covariances of the cluster Gaussian then can be calculated as follows:

$$\begin{aligned} G_m : \mu_m &= E(o_i)_{i=0}^{KN-1} = \frac{1}{kN}\sum_{i=0}^{KN-1} o_i \\ &= \frac{1}{KN}\left(\sum_{k=1}^{K}\sum_{i=(k-1)N}^{kN-1} o_i\right) \\ &= \frac{1}{K}\sum_{k=1}^{K}\frac{1}{N}\sum_{i=(k-1)N}^{kN-1} o_i = \frac{1}{K}\sum_{k=1}^{K}\mu_k \end{aligned} \tag{A.6}$$

$$\begin{aligned} \sigma_m^2 &= E\left((o_i-\mu_m)^2\right)_{i=0}^{KN-1} = E(o_i^2)_{i=0}^{KN-1} - E(\mu_m)^2 \\ &= \frac{1}{KN}\sum_{i=0}^{KN-1} o_i^2 - \mu_m^2 \\ &= \frac{1}{K}\left[\frac{1}{N}\sum_{i=0}^{KN-1} o_i^2 - \sum_{k=1}^{K}\mu_k^2 + \sum_{k=1}^{K}\mu_k^2\right] - \mu_m^2 \\ &= \frac{1}{K}\left[\frac{1}{N}\sum_{k=1}^{K}\left(\sum_{i=(k-1)N}^{kN-1} o_i - N\mu_k^2\right) + \sum_{k=1}^{K}\mu_k^2\right] - \mu_m^2 \\ &= \frac{1}{K}\left[\sum_{k=1}^{K}\left(\frac{1}{N}\sum_{i=(k-1)N}^{kN-1} o_i - \mu_k^2\right) + \sum_{k=1}^{K}\mu_k^2\right] - \mu_m^2 \\ &= \frac{1}{K}\left[\sum_{k=1}^{K}\sigma_k^2 + \sum_{k=1}^{K}\mu_k^2 - K\mu_m^2\right] \end{aligned} \tag{A.7}$$

A.3 A K-means like Algorithm

To partition a set of Gaussians into k clusters a k-means like algorithm can be used. The procedure is as follows:

1. Select k initial centroids of the type belonging to the distance metric that is used (e. g. mean vectors or Gaussians).

2. Repeat steps 3 and 4 until convergence criterion is satisfied (e. g. partitions fixed, centroids fixed, change of average distance measure lower than a given threshold or given number of iterations reached)

3. Determine to which cluster (centroid) each Gaussian belongs. Therefore calculate the distances between each Gaussian and each centroid and assign Gaussian to centroid with smallest value for the distance measure (e. g. weighted Euclidean distance or divergence)

4. For each cluster, calculate a new centroid from the subset of Gaussians which belongs to the cluster. (e. g. new mean vector or approximation of the Gaussians by a single Gaussian (see A.2) depending on the distance metric)

After the algorithm converges, the Gaussians are partitioned in k clusters and for every cluster a centroid exists.

Appendix B

Data Structures

B.1 BBI Trees

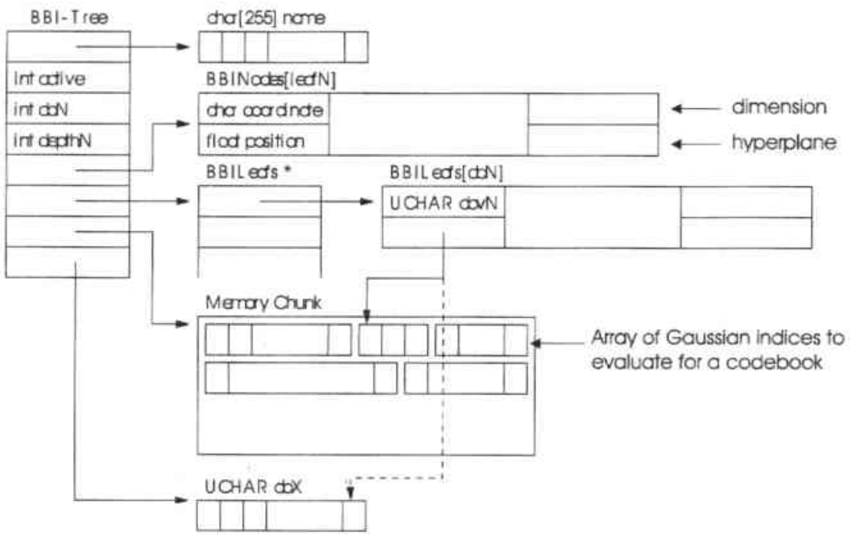


Figure B.1: Data structure of a BBI tree

Since the BBI tree data structure can be implemented in different ways, the implementation used in this work (see figure B.1) will now be described in more detail. As a unique identifier every BBI tree has a name. Furthermore it has a flag, that shows if the BBI tree is activated. To save memory space the nodes and leafs of the tree are stored in arrays, where the structure of the tree is given implicitly. The locations of the nodes and leafs in the arrays are defined by their positions in the tree. Figure B.2 shows the structure of the arrays for a tree of depth 2. The nodes in the nodes array contain the coordinate axis that is divided by the hyperplane belonging to that node and the position where the hyperplane divides this coordinate axis. A second list contains the leafs of the tree, whereas every leaf has a list with entries for every Gaussian mixture. The entries are the number of Gaussians of the Gaussian mixture in the subspace of the leaf and the indices of this Gaussians. All of this indices are stored in larger memory blocks to reduce cache misses. For all Gaussian mixtures with only one Gaussian in a leaf an array containing the indices of the Gaussians is used to avoid the occurrence of duplicate arrays with only one Gaussian index in the memory block.

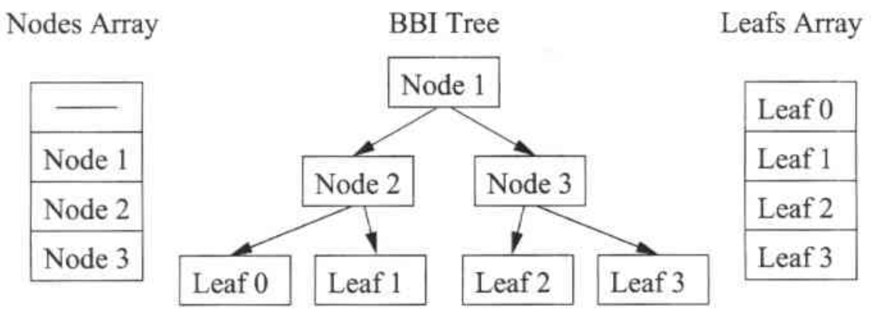


Figure B.2: Example for the storage of the Nodes and Leafs within a BBI trees

B.2 Clustering

When using clustering to speed up the speech recognizer a data structure B.3 is created for each set of clusters. This cluster set data structure can be identified by a unique name. First it contains some variables for handling the data structure. This are the state of activity, the type of the distance measure, the given number of clusters, the number of clusters to be evaluated during decoding, the number of Gaussian mixtures, that belong to this set of clusters, the dimension of the feature space and a list with the numbers of Gaussians assigned to each cluster. Besides that it contains the precalculated value $log((2\pi)^N)$ for a faster evaluation of the cluster Gaussians. If the distance measure is a weighted Euclidean distance, it also contains the average covariance matrix over all Gaussians, that belong to the set of clusters. This covariances can be used as the weight for the weighted Euclidean distance. The centroids of the clusters are stored in two separate sub data structures. All mean vectors for the clusters are stored in a single matrix and if divergence is used as the distance measure, the covariance matrices for the clusters are stored in an additional list. This matrices contain their precalculated determinants for a faster calculation of the cluster Gaussian values. The assignment of the Gaussians to the clusters is specified using a list for each cluster. Each of the lists contains an entry for each Gaussian mixture. This entries specify the indices of the Gaussians within the Gaussian mixture, which belong to this cluster. During the decoding the list of

Gaussians that have to be evaluated, if the cluster belong to the nearest clusters, can be found easily using the index of the Gaussian mixture with this list.

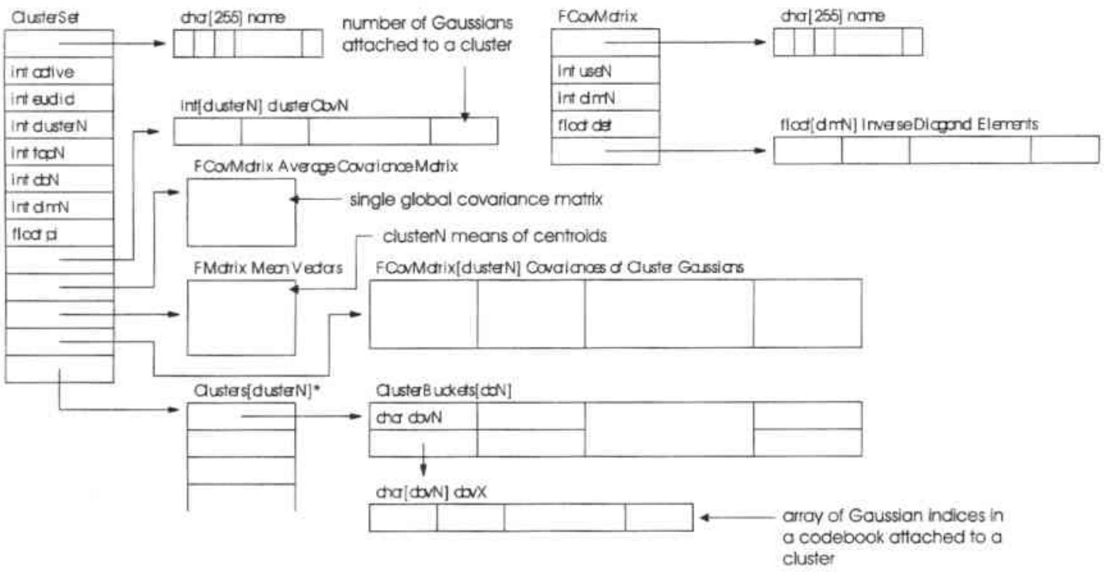

ClusterSet
int active
int eudid
int clusterN
int topN
int cbN
int dimN
float pi
char[255] name
number of Gaussians attached to a cluster
int[clusterN] clusterCbvN
FCovMatrix AverageCovarianceMatrix
single global covariance matrix
clusterN means of centroids
FMatrix MeanVectors
FCovMatrix[clusterN] Covariances of Cluster Gaussians
FCovMatrix
int useN
int dimN
float det
char[255] name
float[dimN] InverseDiagonal Elements
Clusters[clusterN]*
ClusterBuckets[cbN]
char cbvN
char[cbvN] cbvX
array of Gaussian indices in a codebook attached to a cluster

Figure B.3: Data structure of a cluster set

Appendix C

User Interface

C.1 Projection Search

When using the projection search implementation of this thesis, every Codebook has to contain a threshold for the distance of the bounding hyperplanes to the observation vector and the number of dimensions, that have to be bound during the projection search. Since in most cases many codebooks are used, it is easier to use a description file, that specifies the parameters for all codebooks, instead of specifying the parameters for every codebook separately at the start up. The description file consists of three columns, whereas the first contains the name of a codebook, the second contains the number of bounding dimensions for this codebook and the third contains the threshold (see C.1).

Name of codebook	Dimensions to be bound	Distance of hyperplanes
A-b	3	2.0
A-e	3	2.0
A-m	3	2.0
B-b	3	2.0
B-e	3	2.0
B-m	3	2.0
C-b	3	2.0
⋮	⋮	⋮

Table C.1: Example for a projection search description file

The creation of a description file with the same parameters for all codebooks can be done for example with the following TCL-script.

```
# number of bounded dimensions
set depth ''3''
```

```
# threshold for the hyperplanes
set threshold ''2.0''

set descPath ''../desc''
set descFile $descPath/desc.tcl

source $descFile

# ------------------------------------------------------------------
# init modules
# ------------------------------------------------------------------

featureSetInit $SID
codebookSetInit $SID

# creation of the description file
set fp [open ''$descPath/projectionDesc'' w]
# adding of entry for every codebook to the description file
foreach cb [codebookSet$SID] { puts $fp ''$cb $depth $threshold'' }
close $fp
```

To use the projection search, the description file has to be loaded after the initialization of the codebooks during the start up of the decoder. This has to be done using the command:

```
codebookSet$SID readProjectionDesc <filename>
```

In addition to that, the projection search has to be activated.

```
codebookSet$SID set -projOn <active {1,0}>
```

This command also deactivates all other Gaussian selection methods. If the projection search shall not be used during the speaker adaptation, it has to be deactivated during this time using the same command.

C.2 BBI

When using BBI trees as Gaussian selection some precalculation is necessary. At first it has to be specified, which codebook belongs to which BBI tree. Therefore a description file is used. This file consists of two columns. The first column contains the name of a codebook, whereas the second contains the name of the BBI tree it belongs to (see C.2).

Name of codebook	Name of BBI
A-b	OneForAll
A-e	OneForAll
A-m	OneForAll
B-b	OneForAll
B-e	OneForAll
B-m	OneForAll
C-b	OneForAll
⋮	⋮

Table C.2: Example for a BBI description file

When using one BBI tree for all codebooks, this file can be build using a script similar to the one used to create the projection search description file.

```
# name of the BBI tree
set name ''OneForAll''

set descPath ''../desc''
set descFile $descPath/desc.tcl

source $descFile

# ------------------------------------------------------------------
# init modules
# ------------------------------------------------------------------

featureSetInit $SID
codebookSetInit $SID

# creation of the description file
set fp [open ''$descPath/bbiDesc'' w]
# adding of entry for every codebook to the description file
foreach cb [codebookSet$SID] { puts $fp ''$cb OneForAll'' }
close $fp
```

Using this description file the BBI trees have to be build. This has to be done during the preprocessing using for example the following script.

```
# ------------------------------------------------------------------
# Settings for bbi
# ------------------------------------------------------------------
# depth of the BBI trees
set depth ''10''
# threshold for the bounding boxes
set gamma ''0.2''

set descPath ''../desc''
set descFile $descPath/desc.tcl

source $descFile

# -----------------------------------------------------------------
# Init Modules
# -----------------------------------------------------------------

featureSetInit $SID
codebookSetInit $SID

# initialize all BBIs as specified in the description file bbiDesc
bbiSetInit $SID -desc ''bbiDesc''

# -----------------------------------------------------------------
# make bbi trees
# -----------------------------------------------------------------

# build all specified BBI trees with certain depth and threshold
codebookSet$SID makeBBI -depth $depth -gamma $gamma -verbose 1
# save BBI trees
codebookSet$SID saveBBI ''$descPath/bbiTree$SID-$depth-$gamma.gz''

exit
```

To use this precalculated BBI trees during the decoding, the BBI description file and the BBI file have to be loaded during the start up after the initialization of the codebooks. This can be done using:

```
bbiSetInit $SID -desc <bbiDesc> -param <bbiFile>
```

In addition to that the use of the BBIs during the decoding has to be activated:

```
codebookSet$SID set -bbiOn <active {1,0}>
```

This also deactivates the other Gaussian selection methods. If the BBI tree shall not be used during the speaker adaptation, each of the BBI trees has to be deactivated during this time with the following command.

```
codebookSet$SID.bbi(<bbiIndex>) configure -active <active {1,0}>
```

C.3 Clustering

The precalculation, when using clustering as Gaussian selection, is similar to the one of the BBI trees. Again a description file has to be created, which specifies the assignment of each codebook to a set of clusters. The file contains two columns with the names of the codebooks and the names of the according cluster sets. This file can be created using the same script as for the BBI description file for example.

For the creation of the clusters a script similar to the one for making the BBI trees could be used.

```
# output more informations
set verbose 1
# number of iterations of the k-means like algorithm
set iterN 50
# number of clusters in each cluster set
set clusterN 1024
# type of distance measure for the clustering
set euclid 0 {0 = DIVERGENCE, 1 = EUCLID, 2 = WEIGHTED EUCLID}
# type of back-offs
set clsBackOffs 0 {0 = NONE, 1 = ONE GAUSSIAN}

set descPath ''../desc''
set descFile $descPath/desc.tcl

source $descFile

# ----------------------------------------------------------------
# init modules
# ----------------------------------------------------------------

featureSetInit $SID
codebookSetInit $SID

# initialize cluster set as specified
# in the description file clusterSetDesc
clusterSetInit $SID -desc ''clusterSetDesc''
```

```
# ----------------------------------------------------------------
# make cluster sets
# ----------------------------------------------------------------

# make certain number of clusters for each specified set of clusters
# using a certain distance measure
codebookSet$SID makeClusterSet -clusterN $clusterN -iterN $iterN
   -euklid $euclid -verbose $verbose

# make back-offs, if requested and save sets of clusters to file
if {$clsBackOffs == 0} {
 codebookSet$SID saveClusterSet
     ''clusterSet$SID-${clusterN}-${iterN}-e${euclid}.gz''
} else {
 codebookSet$SID makeClusterSetBackOffs
 codebookSet$SID saveClusterSet
     ''clusterSet$SID-${clusterN}-${iterN}-bo-e${euclid}.gz''
}
```

When using Euclidean distance it is also possible to make more difficult assignments, than assigning each Gaussian to the nearest centroid. Therefore two different types of weighted Euclidean distance can be used as distance measure. The two weights are the inverse of the clusters average covariance (assignment = 1) and the inverse of the square root of the product of the clusters average covariance and the centroid covariance (assignment = 2). To create intersecting clusters, the Gaussians are assigned to each centroids, which is closer than a certain threshold. This has to be done before assigning the back-offs with the following command:

```
codebookSet$SID makeClusterSetFinalAssignment
    -assignment <assignment> -threshold <threshold>
```

After creating the clusters, it is possible to use them during the decoding. Therefore the description file and the file with the clusters have to be loaded using:

```
clusterSetInit $SID -desc <clusterDesc> -param <clusterFile>
```

After loading the clusters, the number of nearest clusters that should be used to calculate the score during the decoding has to be specified for every set of clusters.

```
set clsN [codebookSet$SID configure -clsN]
for {set clsX 0} {$clsX < $clsN} {incr clsX} {
 codebookSet$SID.clusterSet($clsX) configure -topN <topN>
}
```

Afterwards the use of the clusters has to be switched on:

```
codebookSet$SID set -clusterOn <active {1,0}>
```

When activating the clustering, the other Gaussians selection methods are deactivated. If the clusters shall not be used during the estimation of the speaker adaptation, they have to be deactivated during this time using the command:

```
codebookSet$SID.clusterSet(<clusterSetIndex>) configure
   -active <active {0,1}>
```

To update the centroids after the speaker adaptation is applied, the following command has to be used:

```
codebookSet$SID updateClusterSetCentroids
```

Bibliography

[AM01] R. Auckenthaler and J. S. Mason. Gaussian Selection Applied to Text-Independent Speaker Verification. Odyssey Speaker Recognition Workshop, June 2001.

[Ben75] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. In *Proc. Commun. Ass. Comput. Mach.*, volume 18(9), pages 509–517, Sept. 1975.

[Boc93] E. Bocchieri. Vector Quantization for the Effizient Computation of Continuous Density Likelihoods. In *Proc. ICASSP*, volume II, pages 692–695, Minneapolis, 1993.

[BU95] P. Beyerlein and M. Ullrich. Hamming Distance Approximation for a Fast Log-Likelihood Computation for Mixture Densities. In *Proc. Eurospeech-1995*, pages 1083–1086, 1995.

[FR96] J. Fritsch and I. Rogina. The Bucket Box Intersection (BBI) Algorithm for Fast Approximative Evaluation of Diagonal Mixture Gaussians. In *Proc. ICASSP '96*, pages 837–840, Atlanta, GA, 1996.

[Fuk90] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition (2nd ed.)*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[Gal97] M. Gales. Maximum likelihood linear transformations for hmm-based speech recognition. Tech. Report CUED/FINFENG/TR291, Cambridge University, 1997.

[Gal99] M. Gales. Semi-Tied Covariance Matrices for Hidden Markov Models. In *Proc. IEEE Transactions Speech and Audio Processing*, volume 7, pages 272–281, 1999.

[GKY99] M. Gales, K. Knill, and S. Young. State-Based Gaussian Selection In Large Vocabulary Continuous Speech Recognition Using HMMs. In *Proc. IEEE Trans. Speech and Audio Processing*, volume 7(2), pages 152–161, 1999.

[HH01] Xuedong Huang and Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001. Foreword By-Raj Reddy.

[HS97] S. M. Herman and R. A. Sukkar. Variable Threshold Vector Quantization for Reduced Continuous Density Likelihood Computation in Speech Recognition. Automatic Speech Recognition and Understanding Workshop, December 14-17 1997.

[LW95] C. J. Leggetter and P. C. Woodland. Maximum Likelihood Linear Regression for Speaker Adaptation of Continuous Density Hidden Markov Models. In *Proc. Computer Speech and Language 9*, pages 171–185, 1995.

[nis] NIST Spoken Language Technology Evaluation and Utility. http://www.nist.gov/speech/tools/index.htm.

[NN96] S. Nene and S. Nayar. Closest Point Search in High Dimensions. In *Proc. CVPR '96*, pages 859–865, 1996.

[OFN97] S. Ortmanns, T. Firzlaff, and H. Ney. Fast Likelihood Computation Methods for Continuous Mixture Densities in Large Vocabulary Speech Recognition. In *Proc. Eurospeech '97*, pages 139–142, Rhodes, Greece, 1997.

[Ort98] S. Ortmanns. *Effiziente Suchverfahren zur Erkennung kontinuirlich gesprochener Sprache.* PhD thesis, Rheinisch-Westfälischen Technischen Hochschule Aachen, Nov. 1998.

[Rab90] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Readings in speech recognition*, pages 267–296, 1990.

[SHH+99] N. Ström, L. Hetherington, T. J. Hazen, E. Sandness, and J. R. Glass. Acoustic Modeling Improvements in a Segment-Based Speech Recognizer. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop*, pages 139–142, Keystone, USA, 1999.

[SMFW01] H. Soltau, F. Metze, C. Fugen, and A. Waibel. A One Pass Decoder Based on Polymorphic Linguistic Context Assignment. In *Proc. Automatic Speech and Recognition Workshop (ASRU)*, Trento, Italy, 2001.

[SPZ05] G. Saon, D. Povey, and G. Zweig. Anatomy of an extremely fast LVCSR decoder. In *Proc. Interspeech 2005 9th European Conference on Speech Communication and Technology September 4-8 2005*, pages 549–552, Lisbon, Portugal, 2005.

[WF97] M. Woszczyna and J. Fritsch. Codebuchübergreifende Bucket-Box-Intersection zur schnellen Berechnung von Emissionswahrscheinlichkeiten im Karlsruher VM-Erkenner. Verbmobil, July 1997.

[WFK$^+$96] A. Waibel, M. Finke, T. Kemp, D. Gates, M. Gavalda;, A. McNair, A. Lavie, L. Levin, L. Mayfield, M. Maier, I. Rogina, K. Shima, T. Sloboda, M. Woszczyna, P. Zhan, and T. Zeppenfeld. JANUS-II — Translation of Spontaneous Conversational Speech. In *Proc. ICASSP '96*, pages 409–412, Atlanta, GA, 1996.

[WSTI95] T. Watanabe, K. Shinoda, K. Takagi, and K. Iso. High Speed Speech Recognition Using Tree-Structured Probability Density Function. In *Proc. ICASSP'95*, volume 1, page 556, Detroit, USA, 1995.