FAST GAUSSIAN EVALUATIONS IN LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION

By

Shivali Srivastava

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

December 2002

Copyright by

Shivali Srivastava

2002

FAST GAUSSIAN EVALUATIONS IN LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION

By

Shivali Srivastava

Approved:

Joseph Picone Professor of Electrical and Computer Engineering (Director of Thesis)	Dr. Georgios Y. Lazarou Assistant Professor of Electrical and Computer Engineering (Committee Member)
Nicolas H. Younan Professor of Electrical and Computer Engineering (Committee Member and Graduate Program Director)	A. Wayne Bennett Dean of the College of Engineering

Name: Shivali Srivastava

Date of Degree: December 13, 2002

Institution: Mississippi State University

Major Field: Electrical Engineering

Major Professor: Dr. Joseph Picone

Title of Study: FAST GAUSSIAN EVALUATIONS IN LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION

Pages in Study: 78

Candidate for Degree of Master of Science

Rapid advances in speech recognition theory, as well as computing hardware, have led to the development of machines that can take human speech as input, decode the information content of the speech, and respond accordingly. Real-time performance of such systems is often dominated by the evaluation of likelihoods in the statistical modeling component of the system. Statistical models are typically implemented using Gaussian mixture distributions.

The primary objective of this thesis was to develop an extension of the Bucket Box Intersection algorithm in which the dimension with the optimal number of splits can be selected when multiple minima are present. The effects of normalization of mixture weights and Gaussian clipping have also been investigated. We show that the Extended BBI algorithm (EBBI) reduces run-time by 21% without introducing any approximation error. EBBI also produced a 12% lower word error rate than Gaussian clipping for the same computational complexity. These approaches were evaluated on a wide variety of tasks including conversational speech.

DEDICATION

I would like to dedicate this thesis to my parents and my husband for their constant support, encouragement and sacrifices throughout my education and career development.

ACKNOWLEDGMENTS

First of all, I want to thank Joe Picone for introducing me to the area of speech recognition. He not only created my interest in this area but also inspired me to do my graduate research in speech recognition. He never got tired of my incessantly argumentative discussions and I always learned something valuable from those discussions. Having him as my mentor for my graduate studies has been a memorable experience for me.

I would also like to thank Jon Hamaker for all the valuable suggestions and ideas he gave me for this research. I owe Jurgen Fritsch for answering any questions I had regarding the Bucket-Box Intersection algorithm which has been used as a baseline for this research.

The environment at ISIP was very conducive to serious research and that has played a big role in my work. Excellent computing resources provided in the lab helped me a lot with the experiments. My years at ISIP will always be a sweet memory for me not only for the knowledge I gained there but also for giving me an opportunity to meet some very wonderful people. I deeply thank all my colleagues there for making those years so enjoyable in spite of the heavy work load we always had.

Finally much thanks to my wonderful husband Hamza, without his love and support this research would not have been successful.

TABLE OF CONTENTS

DEDICATION i

ACKNOWLEDGMENTS ii

LIST OF TABLES v

LIST OF FIGURES vii

CHAPTER

I. INTRODUCTION 1

1.1 A Statistical Approach to Speech Recognition 3
1.2 Resource Requirements 6
1.3 Fast Gaussian Evaluations 8
1.4 Thesis Contributions 9
1.5 Structure of the Thesis 11

II. BUCKET BOX INTERSECTION 12

2.1 K-Dimensional Binary Search Tree 12
2.2 Building a K-D Tree for the HMM-Based Recognition System 13
2.3 Optimization of a K-D Tree 17
2.4 Approximation Error Introduced by the K-D Tree 22

III. EXTENDED BUCKET BOX INTERSECTION ALGORITHM 24

3.1 A Modified Optimization Criterion for Multiple Minima 24
3.2 Mixture Weight Re-Normalization 29
3.3 Selection of a K-D Tree 33

IV. EXPERIMENTS 36

4.1 Experimental Databases 36

TIDIGITS 36
Alphadigits 37
SWITCHBOARD 37
4.2 Measurement of Performance 38
4.3 Profiling Tool 38
4.4 Effects of Algorithm Parameters on the Algorithm Performance 39
Effect of Tree Depth on Speedup and Approximation Error 39
Effect of Threshold on Speedup and Approximation Error 39
4.5 Relative Threshold 40
4.6 Experimental Results and Analysis 41
TIDIGITS 41
Alphadigits 46
SWITCHBOARD 60
4.7 Gaussian Clipping and Extended BBI Algorithm 60

V. CONCLUSIONS AND FUTURE DIRECTIONS 66

5.1 Thesis Contributions 66
Extended BBI Algorithm and BBI Algorithm 66
Gaussian Clipping and Extended BBI Algorithm 67
5.2 Summary of Experiments 67
5.3 Future Work 68

REFERENCES 69

APPENDIX A 77

LIST OF TABLES

1. The effect of the vocabulary size and perplexity of various tasks on the performance of the ASR system 2

2. The performance of the Alphadigits system using the BBI and Extended BBI algorithms 29

3. The performance of the Alphadigits system using the BBI and Extended BBI algorithms, where the Extended BBI algorithm is using re-normalized mixture weights. 32

4. The performance of the Alphadigits system using the BBI and Extended BBI algorithms. The BBI system is using a single big tree shared by all the states and the Extended BBI system is using one k-d tree per state 34

5. The performance of the BBI and Extended BBI algorithms, where the Extended BBI algorithm is using state level k-d trees, a modified optimization criterion and mixture weight re-normalization 34

6. Performance of the TIDIGITS system using the Extended BBI algorithm . 42

7. Performance of the Alphadigits system using the Extended BBI algorithm. The system used 16-mixture components 47

8. Percentage speedup in Gaussian evaluation and WER for several test runs with different values of the algorithm parameters for the Alphadigits system with 32-mixture components 51

9. Performance of the Alphadigits system using the Extended BBI algorithm. The system used 64-mixture components 56

10. The approximation error introduced by the Extended BBI algorithm to obtain a certain Gaussian speedup for the Alphadigits system with varying mixture components 59

11. The performance of the Extended BBI algorithm using different values of tree depth and relative threshold for a SWB task 61

12. The performance of the system using fewer mixture components and no Gaussian speedup techniques, the system using the Extended BBI algorithm and the system using the Gaussian clipping algorithm 64

13. Execution time of the Gaussian computations for several test runs 78

LIST OF FIGURES

1. Memory and the CPU time used by various parts of a current state-of-the-art speech recognition system 6

2. The effect of an increase in the number of mixture components on the CPU time used by the Gaussian evaluations 7

3. An example of building a k-d tree in a 2-dimensional region 14

4. The k-d tree of depth 4 corresponding to the regions divided by the hyperplanes H_1 through H_{15} in Figure 3 15

5. A single multivariate Gaussian in a 3-dimensional space. The elliptical region represents a hyper-ellipsoid region of the Gaussian 16

6. A Gaussian box corresponding to the hyper-ellipsoid region in Figure 5 . . 16

7. A study of the number of Gaussians in the left and right regions with respect to the hyperplane position along the coordinate axis $j_{0,0}$ 20

8. Behavior of the variables N_l and N_r with respect to the hyperplane $h_{0,0}$. Note that the projection boundaries of the Gaussians are shown to be evenly distributed along the horizontal axis in this figure 21

9. An example of multiple minima in the number of Gaussian splits. The case 1 contains a hyperplane A_1B_1 orthogonal to the axis with a lower variance and the case 2 contains a hyperplane AB orthogonal to the axis with a higher variance 25

10. An example of multiple minima in the number of Gaussian split. The case 1 uses separating hyperplanes which are orthogonal to the axis with a lower variance and the case 2 uses the hyperplane which is orthogonal to the axis with a higher variance 27

11. The effect of variations in relative threshold on the speedup in Gaussian evaluations for a TIDIGIT system with a constant k-d tree depth. The k-d trees with depth 3, 4 and 5 are used 43

FIGURE Page

12. The effect of variations in the relative threshold on the WER of a TIDIGIT system with a constant k-d tree depth 44

13. The effect of the k-d tree depth and threshold on the WER and Gaussian speedup for the TIDIGITS system using the Extended BBI algorithm 45

14. The variation in Gaussian speedup as a function of relative threshold for the Alphadigits system with 16-mixture components. The plots correspond to the tree depths of 3, 4 and 5 48

15. The effect of variations in relative threshold on the WER of the Alphadigits system with 16-mixture components for a constant tree depth 49

16. The effect of variations in tree depth and relative threshold on the WER and Gaussian speedup for the Alphadigits system using 16-mixture cross-word triphone models. 50

17. The variations in the Gaussian speedup as a function of the relative threshold and k-d tree depth for the Alphadigits system using 32-mixture cross-word triphone models 52

18. The effect of the algorithm parameters on the WER of the Alphadigits system using 32-mixture components 53

19. The simultaneous effect of the algorithm parameters on the Gaussian speedup and WER of the Alphadigits system using 32-mixture components. Different points on the plots give the Gaussian speedup and WER for a value of the tree depth and threshold 54

20. The effect of the algorithm parameters on the speedup in Gaussian evaluations for the Alphadigits system using 64-mixture components 57

21. The variations in WER as a function of the algorithm parameters for the Alphadigits system using 64-mixture components 58

22. The effect of algorithm parameters on the Gaussian speedup and WER for the Alphadigits system using 64-mixture components 59

23. The speedup in Gaussian evaluations as a function of the relative threshold and k-d tree depth for a SWB system 62

24. The effect of the algorithm parameters on the WER for a SWB system . . . 63

25. The effect of the algorithm parameters on the Gaussian speedup and WER for the SWB task 64

CHAPTER 1

INTRODUCTION

Speech recognition technology has made a significant progress from the days of isolated word recognition. Five decades of interdisciplinary research in widely different areas, such as, linguistics, psychacoustics, signal processing, computer science, pattern recognition, and information theory, has greatly advanced the state of the art in speech recognition systems. Rapid advances in speech recognition theory, as well as computing hardware, have led to the development of machines that can take human speech as input, decode the information content of the speech, and respond accordingly [1]. This has greatly increased the range of applications for automatic speech recognition (ASR) technology. However, these applications require large vocabulary continuous speech recognition (LVCSR) systems with negligibly small amounts of latency.

The primary evaluation criterion for speech recognition research is the word error rate (WER). Though the WER on standard evaluation tasks has decreased consistently over the years, the complexity of the evaluation task (and the associated speech recognition systems) has increased significantly. Task complexity can be measured in terms of perplexity [1]. The perplexity is defined as:

$$Perplexity = 2^{H}, \tag{1}$$

where H is the entropy of the language model in bits [1]. Table 1 provides a comparison of perplexity and WER for a variety of tasks [2]. A relationship between the word error rate of the system and the perplexity of a task is approximated by the following equation [2]:

$$WER = -12.37 + 6.42 \cdot \log(Perplexity) \quad (2)$$

While research has focused on decreasing WER, very few investigations of the trade-off between WER and recognition speed have been performed [3, 4]. The recognition speed is defined as the required CPU time measured in seconds to decode one second of input speech [5]. Real time factor is used as the unit for recognition speed. For example, a speed of NxRT means that the computer takes N seconds to decode an utterance that is one second long, where the decoding time is measured in CPU seconds.

The need for real-time LVCSR systems has created a growing interest in developing algorithms for faster recognition. Many such algorithms and techniques include elimination of and/or approximations for many computationally expensive

Corpus	Vocabulary Size	Perplexity	WER
TI DIGIT	11	11	~0.0%
OGI Alphadigit	36	36	8.0%
Resource Management (RM)	1,000	60	4.0%
Air Travel Information Service	1,800	12	4.0%
Wall Street Journal	20,000	200-250	15.0%
Broadcast News	>80,000	200-250	20.0%
Conversational Speech	>50,000	100-150	30.0%

Table 1. The effect of the vocabulary size and perplexity of various tasks on the performance of an ASR system.

components. These approximate techniques increase the WER of the system. However, to convert laboratory systems into useful products, we must develop more efficient systems [6]. To spur progress in this direction, the Defense Advanced Research Projects Agency (DARPA) and the National Institute of Standards and Technology (NIST) conducted evaluations on Broadcast News (Hub 4) that included a test investigating the performance of the systems running under 10xRT on a single processor [7, 79]. BBN Technologies showed that elimination and approximation of various computations produced a speedup by a factor of 20 with a relative loss in WER of 18% [8].

In the Rich Transcription Evaluation (RT-02) (also conducted by DARPA and NIST), Cambridge University's Hidden Markov Model toolkit (HTK) used a faster contrast system (cu-htk2) with a simpler architecture. This system didn't use triphone and quinphone rescoring. On the 2002 evaluation set, this system reduced the run-time from 320xRT to 67xRT but increased the WER from 23.9% to 26.7% [9]. A faster version of the full HTK 2002 system was also developed which ran in less than 10xRT. On the same 2002 evaluation set, the performance of this system was only 0.5% (absolute) worse than the cu-htk2 system [9]. A major focus of this thesis is to develop methods to increase recognition speed without significantly increasing WER.

1.1. A Statistical Approach to Speech Recognition

In statistical terms, the task of an ASR system is to find the most likely word sequence, W, given the acoustic evidence, A [10]. Mathematically, the recognizer chooses a word string W, which satisfies [27]:

$$\hat{W} = \underset{W}{\text{argmax}}\ p(W/A) \tag{3}$$

By using a Bayesian approach [11], the problem can be simplified to:

$$\hat{W} = \underset{W}{\text{argmax}}\ p(A/W)p(W) \tag{4}$$

The above formula determines the design of the speech recognizer. The probability $p(A/W)$ is computed using an *acoustic model* [11-16], while the estimate of the probability, $p(W)$, is determined using a *language model* [17]. The recognizer combines the acoustic and language model probabilities to form the probability of word sequences (e.g., sentences). The recognizer's main task is to search over all possible word strings to find the most probable word sequence [10, 11], a process referred to as decoding [18-22]. The acoustic front end extracts features from the speech signal which capture the temporal and spectral characteristics of a signal. A detailed tutorial on acoustic front end can be found in [23-26].

In current speech recognition systems, hidden Markov models (HMMs) are the basic building blocks of the acoustic model [10, 23]. Most LVCSR systems model phonetic variability using HMMs, and consider the context of a phone as part of the model. For example, a phone *ph* will be modeled as a context-dependent phone [31], referred to as a triphone, of the form ***a-ph+b***, where *a* and *b* define the left and right contexts. Typically, LVCSR systems use triphones, though more powerful systems use longer acoustic context (e.g., quinphone). An HMM consists of a Markov distribution [19, 33, 34, 35] for transitions across different states, and includes a probability density

function at each state that models the probability of the output symbols possible at that state. The choice of output probability function is crucial as it must model all of the intrinsic spectral variability in the speech signal [36]. Typically, a mixture of multivariate Gaussian distributions is used for this output distribution model. Each triphone model corresponds to an elementary HMM with starting and stopping states. Systems use a 5-state HMM [37, 38] for each triphone, which includes a dummy start state, a dummy stop state and three information-bearing states.

In the computation of $p(A/W)$, it is necessary to compute the probability density that a feature vector was generated by that state. This probability density function, which is commonly modeled by a mixture of multivariate Gaussian distributions, is assumed to depend only on the current feature vector and state, and can be written as [10, 41, 42]:

$$p(x;s(\mu, \Sigma)) = \sum_{m=1}^{M} w_{jm} \cdot \frac{1}{\sqrt{(2\pi)^k |\Sigma_{jm}|}} \exp\left(-\frac{1}{2}(x_j - \mu_{jm})^T \Sigma_{jm}^{-1}(x_j - \mu_{jm})\right) \quad (5)$$

under the assumption that the vector $x_j \in R_k$, where the region R_k is mathematically represented as $R_k = \{\boldsymbol{x}: -\infty < x_i < \infty, \; for \; i = 1, 2, \ldots, k\}$.

In Equation 5,

- w_{jm} — the weight for the m^{th} Gaussian of the j^h state
- x_j — the feature vector
- $s(\mu, \Sigma)$ — the state of HMM having multiple Gaussians
- μ_{jm} — the mean vector for m^{th} Gaussian of the j^h state

- Σ_{jm} — the variance-covariance matrix for m^{th} Gaussian of the j^{h} state
- M — the number of Gaussians in the state

The distance between the feature vector and the mean of the Gaussian is modeled using a likelihood measure that is computed by taking the log of Equation 5. This distance metric is known as the Mahalanobis distance [40].

1.2. Resource Requirements

Speech recognition is a resource intensive task. The percentage of the total CPU time and memory used by various parts of a current state-of-the-art ASR system is shown in Figure 1. The largest percentage of the CPU time is used in acoustic modeling, while the largest percentage of memory is devoted to search. In general, most current state-of-the-art LVCSR systems typically spend about 50% to 80% of the total CPU time on the computation of observation probabilities with mixtures of multivariate Gaussians [28]. This time depends on various factors, which include the size of the

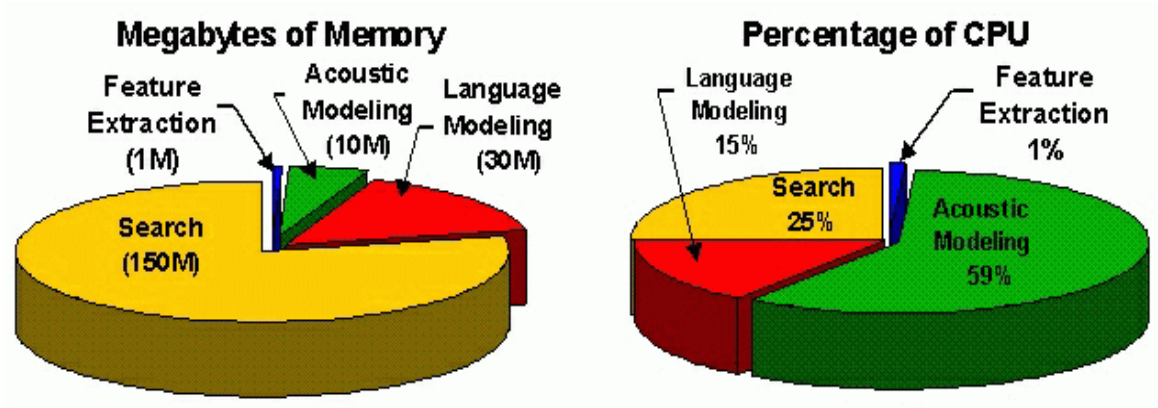

Figure 1. Memory and the CPU time used by various parts of a current state-of-the-art speech recognition system.

vocabulary, the complexity of the acoustic models and the distance measure used during Gaussian evaluations.

The CPU time consumed by the Gaussian evaluations increases with the number of mixtures used to model context-dependent phones. Figure 2 shows that the percentage of the CPU time used in Gaussian evaluations increases approximately by a factor of three as the number of mixtures increases from 8 to 64 for the Alphadigits task. Similar experiments showed that the percentage of the CPU time used for Gaussian computations increases by a factor of three on the SWITCHBOARD task as the number of Gaussian mixture components in a HMM state varies from 12 to 16 [29]. On the same task, the CPU load for the Gaussian computations for a network decoder increased by a factor of five as the number of Gaussians in a HMM state increases from 2 to 12. Similarly, the use of quinphone or other longer time-span acoustic models considerably increases the amount of time required for the computation of observation probabilities.

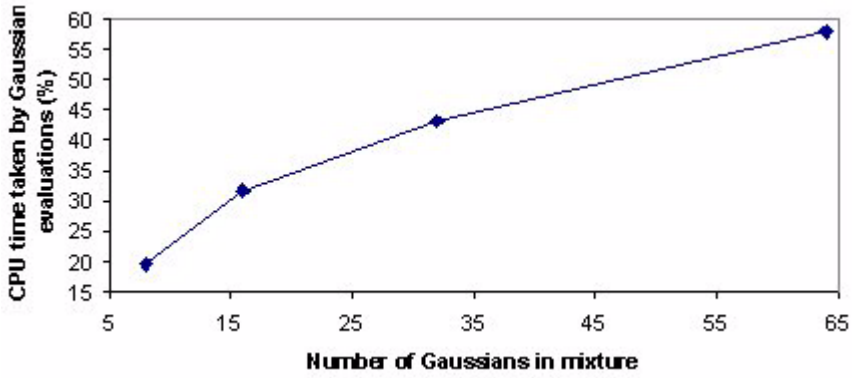

Figure 2. The effect of an increase in the number of mixture components on the CPU time used by Gaussian evaluations.

1.3. Fast Gaussian Evaluations

We have established that computations in real systems are dominated by Gaussian evaluations. Therefore, in order to reduce the run-time of the system, it is necessary to reduce the time consumed by Gaussian evaluations without increasing the system WER. We begin with a simple observation: if a feature vector lies on the tail of a distribution, then the likelihood of that distribution producing that feature vector is very small [6]. Such an input vector is known to be an outlier with respect to that Gaussian distribution. Thus, removing these Gaussians from the computation of log probability for this feature vector does not produce a significant degradation in the accuracy of the distance computed.

This was the motivation for an approach known as *Gaussian Selection* [43, 44]. Using this technique, a speedup in the likelihood computations ranging from 3x to 9x was reported [43]. The most important step in Gaussian selection is to find a method which can efficiently find the Gaussians which do not make a significant contribution to the overall mixture probability. Vector quantization (VQ) like approaches were successfully used to find these Gaussians and approximate the log probability computations [45-49].

Another technique that is most commonly used is the Nearest Neighbor Approximation [50-52], which uses the Gaussian with the smallest Mahalanobis distance to the feature vector at a particular frame for the likelihood computation. The nearest neighbor approach efficiently determines the Gaussian closest to the input vector among all the Gaussians in the k-dimensional space. Therefore, in this approach we do not compute a sum of Gaussians [39, 50]. The computation can be kept completely in the logarithmic space, thereby reducing the computational complexity.

Several VQ based approaches require partial computation of a Mahalanobis distance for each Gaussian in the mixture distribution and require scanning all the Gaussians to find the most significant Gaussians [46]. This greatly increases the computational load. J. Fritsch *et al* proposed the Bucket Voronoi Intersection (BVI) [53] and Bucket Box Intersection (BBI) [54] algorithms for fast search. These algorithms use a tree-based search to find the most significant Gaussians. In order to build the tree, the feature space is partitioned into several cuboids with edges parallel to the coordinate axes. These cuboids divide the feature space into several *voronoi* regions. These voronoi regions are used to build a tree to search the most significant Gaussian. This thesis is based on this latter work.

1.4. Thesis Contributions

The primary objective of this thesis was to investigate the BBI algorithm and develop an Extended BBI (EBBI) algorithm. Though the BBI algorithm has enjoyed significant success, an extensive study of the algorithm suggested the possibility of improvements. Several issues specific to the BBI algorithm have been addressed in this thesis:

- **Modifications of the optimization criterion to deal with multiple minima:** Previous implementations of the BBI algorithm [39, 54] did not consider the case when multiple minima in the number of splits occur during the optimization process. We propose a modified optimization criterion in which the dimension with the optimal number of splits can be selected in this case.
- **Normalization of mixture weights:** The effect of using the normalized weights during the Gaussian score computations has been studied in a great detail.

- **Gaussian clipping:** Generation of a lower-order mixture model from a higher-order model by clipping the Gaussians with small mixture weights is known as Gaussian clipping. Gaussian clipping has been introduced in this thesis to speed up computations. Mixture weights have been renormalized to insure that each model disregards only a certain percentage of error.

- **State level trees:** A state-level binary tree was used in this thesis to obtain an improvement in speed. In general, there are several possibilities:

  (1) Use a single large binary tree shared by all HMM states.

  (2) Let several HMM states share a binary tree (phonological tying, such as that used in phonetic decision trees can be used) [28].

  (3) Let each HMM state use one tree.

  The optimal choice depends upon the task, available memory (to store trees) and other such parameters.

The first approach produces a slightly greater speedup than the third approach, but introduces a higher degradation in the performance of the system. The second approach produces the maximum speed up and requires the least memory, but introduces a larger increases in the WER. The use of a state-level tree introduces the least approximation error and results in the significant speedups compared to the first approach. Since the objective of this thesis is to develop a technique to speed up score computations while maintaining accuracy, the state-level tree approach has been used.

Hence, we show that Gaussian speedup reduces the total run-time (RT) of an LVCSR system. We have developed the Extended BBI algorithm to achieve a maximum improvement of 21% in the run-time of the system without introducing any approximation error. This is significant because we are not increasing the WER of the system to achieve this speedup. This is described in more detail in Chapter 4.

1.5. Structure of the Thesis

In Chapter 2, we provide a detailed mathematical overview of the BBI algorithm. This chapter explains the search criterion used by the BBI algorithm and discusses the optimization of this search criterion. Chapter 3 discusses the problems with previous implementations of the BBI algorithm and proposes an improved BBI algorithm known as Extended BBI. This chapter also includes the experimental results which verify the superiority of the Extended BBI algorithm. In Chapter 4, we present the experimental design and present the experimental results for the proposed algorithm. The experimental results presented in Chapter 3 compare the performance of the BBI and Extended BBI algorithms. The experimental results presented in Chapter 4 provide an understanding of the performance of the Extended BBI algorithm with respect to the variations in algorithm parameters and the number of Gaussians in a mixture distribution. In Chapter 5, we summarize the major findings in this work and discuss promising future directions.

CHAPTER 2

BUCKET BOX INTERSECTION

This chapter presents an overview of the Bucket Box Intersection algorithm used for fast Gaussian evaluations, which includes searching, training, and optimization. Special emphasis is placed on the optimization criterion used to minimize the search complexity.

2.1. K-Dimensional Binary Search Tree

We have discussed that the state log-likelihood of an input vector can be evaluated by using the dominant Gaussian components, without significantly increasing the WER of the system. This approach needs an efficient way to select such Gaussian components. The data structure which allows for fast search in the BBI algorithm is known as a k-d tree [55], where *k* is the dimensionality of the feature space.

A k-d tree is a k-dimensional space partitioning tree. It is an efficient data structure with respect to storage. The average search time for a k-d tree is $O(\log n)$, where n is the number of records in the file. At every non-terminal node of the tree, the current k-dimensional region, R^K, is divided into two half spaces by means of a hyperplane orthogonal to one of the *k* coordinate axes [58]. The hyperplane is represented as a pair of

two quantities — j, the index to the coordinate axis orthogonal to the plane and h, the location of the plane on this axis [59]. This is mathematically represented as $H = (j, h)$. The initial region represents the root node. The two regions obtained by splitting the root region are called the *left* and *right* child of the root node and are represented as $l_{child} = \{x \in R^K : x_j \leq h\}$ and $r_{child} = \{x \in R^K : x_j > h\}$. Any feature vector can be located with respect to the hyperplane by a single scalar comparison of the j^{th} component of a feature vector with the location of the hyperplane h. A k-d tree of depth *d* partitions a k-dimensional space in 2^d disjoint regions called *buckets* of the tree.

Figure 3 shows an example region $abcd$ in a 2-dimensional space. The region $abcd$ represents the root node of the k-d tree and contains a hyperplane H_1 orthogonal to the axis X_1. This hyperplane splits the region $abcd$ into two regions, $abef$ and $ecdf$. These regions are further divided by the hyperplanes H_2 and H_3, both orthogonal to the coordinate axis X_2. Successive splits by hyperplanes H_4 through H_{15}, as shown in the Figure 3, build a k-d tree of depth 4. The resulting k-d tree is shown in Figure 4. A feature vector $f_v = (x_1, x_2)$ in the region $abcd$ can be located in this framework by a sequence of four scalar comparisons, which lead to the bucket $ukhv$.

2.2. Building a K-D Tree for the HMM-Based Recognition System

This section describes the procedure for building a k-d tree for the HMM based speech recognition system [39, 53, 54]. Let us consider a single multivariate Gaussian in a

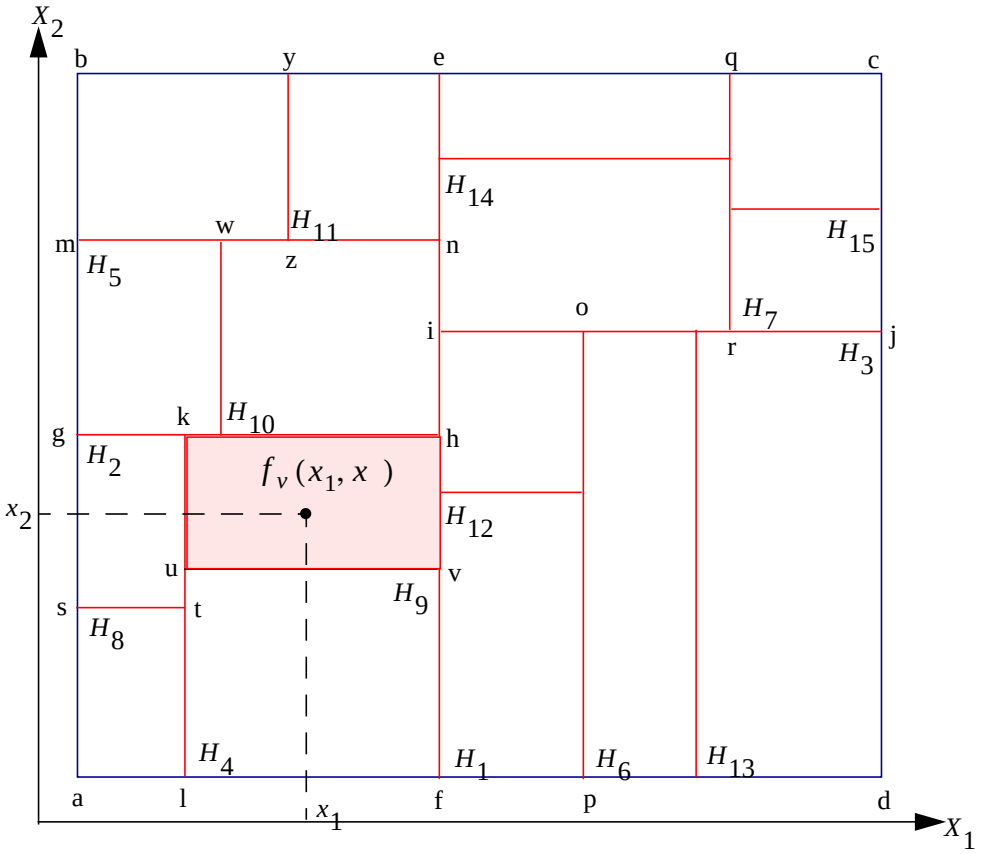

Figure 3. An example of building a k-d tree in a 2-dimensional region.

k-dimensional space with a mean vector $\mu = [\mu_1, \mu_2, \ldots, \mu_k]$ and a covariance matrix $\Sigma = (\sigma_1^2, \sigma_2^2, \ldots, \sigma_k^2)$. Assuming a diagonal covariance matrix, the log-likelihood of this Gaussian is:

$$\log P(x;s(\mu, \Sigma)) = -\frac{1}{2}\left[\log\left\{(2\pi)^k \prod_{i=1}^{k} \sigma_i^2\right\} + \sum_{i=1}^{k} \frac{(x_i - \mu_i)^2}{\sigma_i^2}\right] \quad (6)$$

The region in a *k*-dimensional space, where the log-probability given by Equation 6 is greater than an absolute threshold, *T*, defines a hyper-ellipsoid with axes parallel to the coordinate axes [39]. Using this absolute threshold value, a

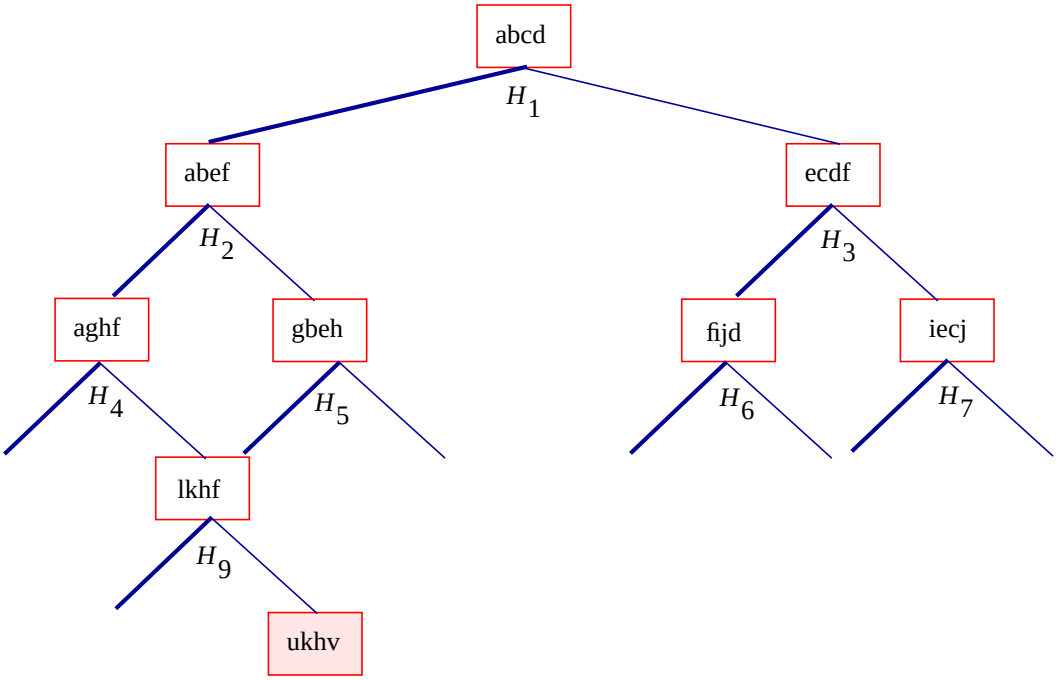

Figure 4. A k-d tree of depth 4 corresponding to the regions divided by the hyperplanes H_1 through H_{15} in Figure 3.

multidimensional box can be computed which completely encloses the hyper-ellipsoid region. This box is known as a *Gaussian box* [54]. These boxes are used to build a k-d tree. Figure 5 represents an example of a Gaussian in a 3-dimensional space and a corresponding hyper-ellipsoid region. Figure 6 represents a Gaussian box which includes the hyper-ellipsoid region of Gaussian. The values x_L, x_U, y_L and y_U are the lower and upper projection boundaries of the Gaussian box along X and Y coordinates.

The projection boundaries can be calculated for each coordinate axis by using the specified threshold value [54]. Substituting T for the log probability in Equation 6 yields,

$$T = -\frac{1}{2}\left[\log\left\{(2\pi)^k \prod_{i=1}^{k} \sigma_i^2\right\} + \sum_{i=1}^{k} \frac{(x_i - \mu_i)^2}{\sigma_i^2}\right]. \quad (7)$$

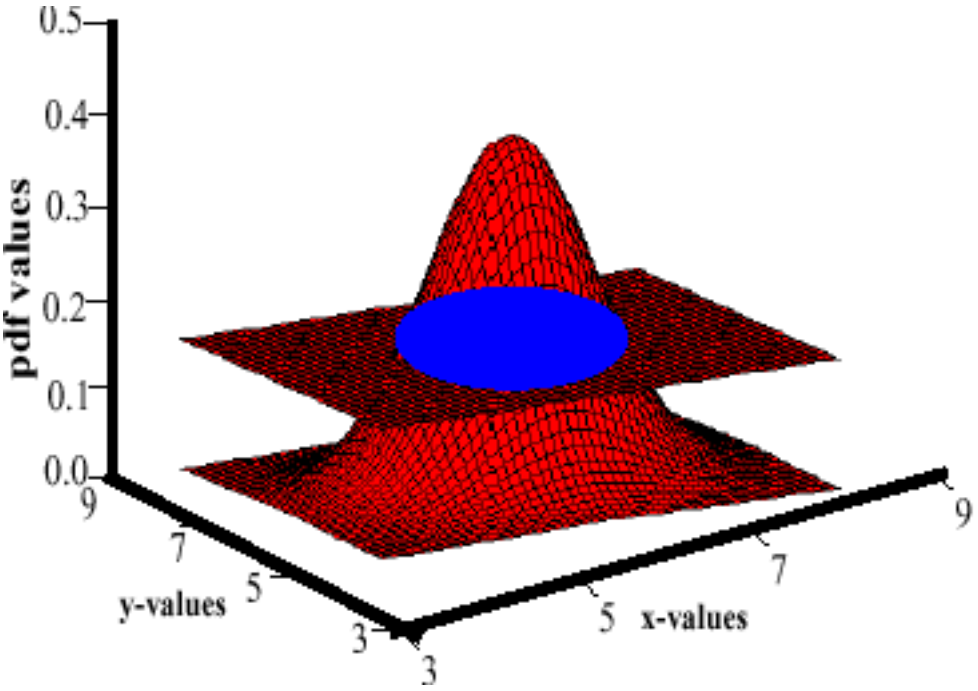

Figure 5. A single multivariate Gaussian in a 3-dimensional space. The surface that results from intersecting the Gaussian with a plane can be represented by a hyper-ellipsoid region of the Gaussian.

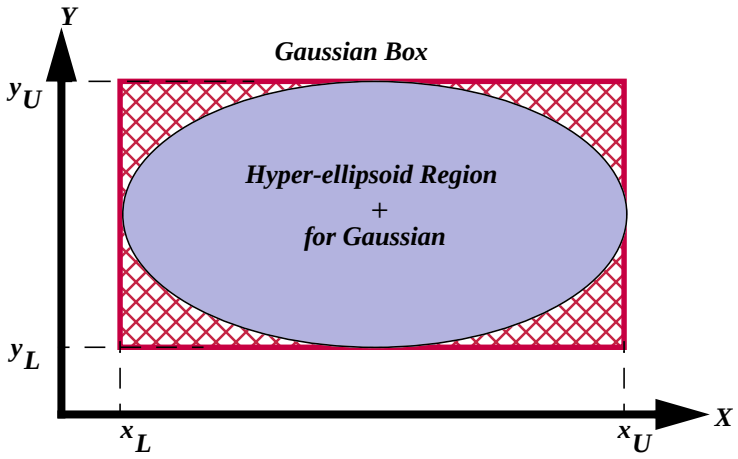

Figure 6. A Gaussian box corresponding to the hyper-ellipsoid region in Figure 5.

Solving Equation 7 for x_i gives the lower and upper projection boundaries of a Gaussian box for the coordinate axis i as follows:

$$x_i = \mu_i \pm \sqrt{-2\sigma_i^2 \left[T + \frac{1}{2}\log\left\{ (2\pi)^k \prod_{i=1}^{k} \sigma_i^2 \right\} \right]}. \quad (8)$$

2.3. Optimization of a K-D Tree

Let us consider a k-d tree of depth *d* with N Gaussians in the initial root region. Given that the feature vector f_v belongs to the root region, the cost of the search is $S_C(root) = N$. Under the k-d tree framework, if $f_v \in Bucket_i$, the cost of the search for the bucket $Bucket_i$ is $S_C(Bucket_i) = N_i$. Thus, the cost of the search using the k-d tree framework reduces from N to N_i. If p_i is the probability that the feature vector lies in the i^{th} bucket and N_i is the number of Gaussians whose boxes intersect with this bucket, the average cost of the search for the i^{th} bucket can be written as [60, 61].

$$Avg(S_c(Bucket_i)) = p_i \cdot N_i \quad (9)$$

The expected cost of search for the tree is:

$$Exp(S_C) = \sum_{i=1}^{N_{buckets}} p_i \cdot N_i \quad (10)$$

The objective of the k-d tree optimization is to minimize the expected cost of search. This results in a global optimization criterion (GOC) [60], which involves the

variables p_i and N_i. Since these variables are functions of the division hyperplane at each non-terminal node of the tree, the GOC requires a joint optimization of the choice of the division hyperplane at each non-terminal node of the k-d tree. This is an extremely complex and practically unfeasible criterion. Thus, it becomes necessary to locally optimize the tree to achieve the minimum expected search time. This can be achieved by independently minimizing the expected search at each node. This results in a local optimization criterion (LOC) [61].

Let us consider a bounded region R_b with N_{R_b} Gaussians that have a non-empty intersection. Let p_l and p_r be the probabilities that a feature vector lies in the l_{region} and r_{region} respectively. Let N_l and N_r be the number of Gaussians whose boxes have a non-empty intersection with the left and right regions respectively. The variables p_l, p_r, N_l and N_r are functions of the division hyperplane [59] parameters $j_{0,0}$ and $h_{0,0}$. An ideal division, producing the best possible partition to achieve the minimum search complexity, is the one which reduces the search complexity to half [62]. Thus, for an ideal division, $p_l = p_r = 1/2$ and $N_l = N_r = N_{R_b}/2$. In other words, for an optimal division, $N_l(h) - N_r(h) = 0$ [63].

If N_{left} and N_{right} are the number of Gaussians whose boxes are entirely in the left and right regions respectively and N_{split} is the number of Gaussians whose boxes are split by the hyperplane, then $N_l = N_{left} + N_{split}$ and $N_r = N_{right} + N_{split}$, with the

condition $N_{left} + N_{right} + N_{split} = N_{R_b}$ [63]. A hyperplane for which $N_{split} = 0$,will give minimum values of N_l and N_r.

The functional dependence of the variables p_l and p_r on the division hyperplane is determined by the distribution of the feature vectors in the initial region and the Gaussian boxes which intersect with that region [59]. The root region contains N_{R_b} Gaussians. The projections of these Gaussians on the coordinate axis $j_{0,0}$ gives $2 \cdot N_{R_b}$ projection boundaries along that axis. Let us assume that $(l_1, l_2, \ldots, l_{2 \cdot N_{R_b}})$ are the locations of the projection boundaries along the coordinate axis in a sorted order. If the hyperplane position is such that $h_{0,0} = L_j + \Delta$, where Δ is a small positive value and $h_{0,0} \leq l_1$, then $N_l(h_{0,0}) = 0$ and $N_r(h_{0,0}) = N_{R_b}$. If we slowly vary the hyperplane position along the axis from L_j to U_j, the values of N_l and N_r change as follows [63, 64]:

- When the hyperplane crosses a lower projection boundary, a new Gaussian gets included in the left region and no Gaussian gets excluded from the right region. Thus, N_l increases by 1 and N_r remains unaltered.

- When the hyperplane crosses an upper projection boundary, one Gaussian gets excluded from the right region and no new Gaussian gets included in the left region. Thus, N_l remains unaltered but N_r decreases by 1.

- If the position of $h_{0,0}$ varies between two adjacent projection boundaries, the number of Gaussians in both the regions remain same. Thus, the values of N_l and N_r remain constant.

Let us consider the example given in Figure 7. A hyperplane $h_{0,0}$ is moved along the axis from the lowest boundary L_j towards the highest boundary U_j. There are nine different positions of the hyperplane along the axis. Figure 8 shows the behavior of N_l and N_r as a function of $h_{0,0}$. The projection boundaries in Figure 7 are indicated as $(1, 2, \ldots, 2 \cdot N_{R_b})$, with $N_{R_b} = 8$. It can be observed that when the hyperplane position is such that $\boldsymbol{h_{0,0} \in (l_8, l_9)}$, the values of N_l and N_r are equal. Such a hyperplane results in

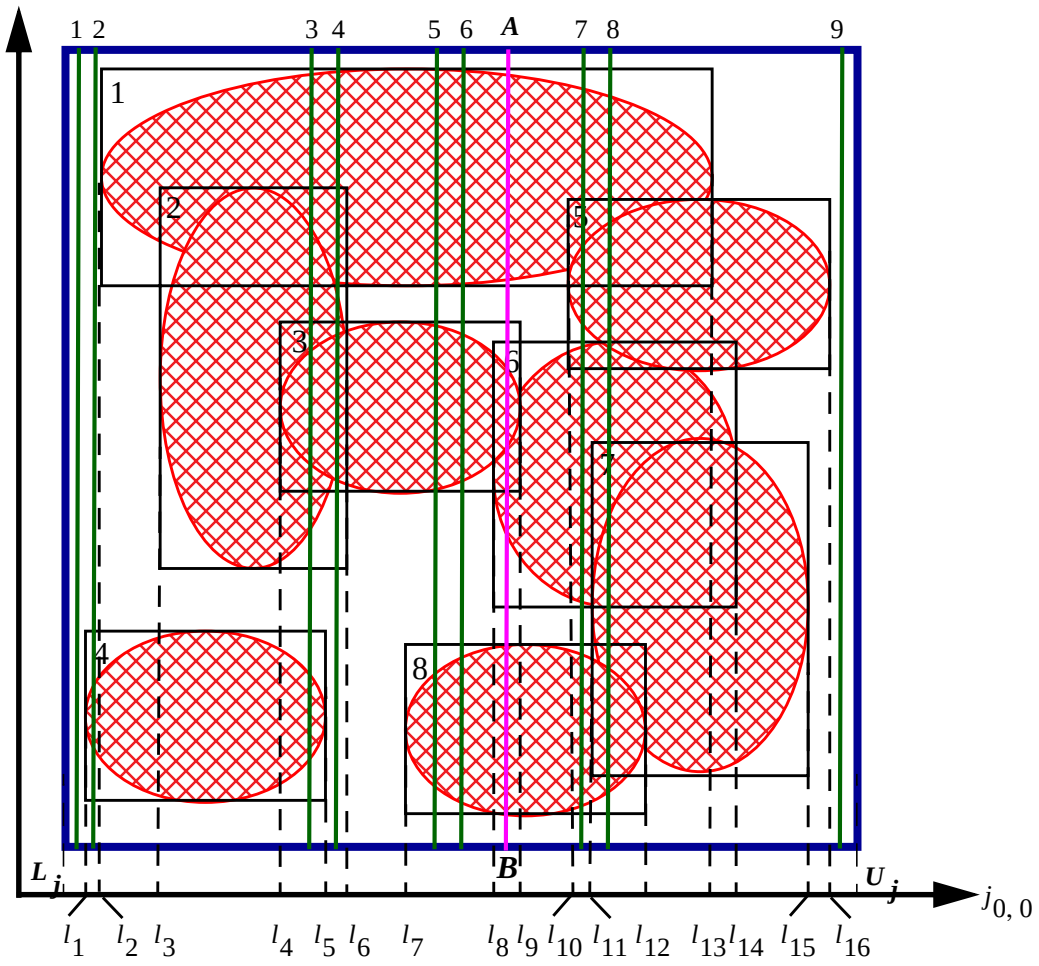

Figure 7. A study of the number of Gaussians in the left and right regions with respect to the hyperplane position along the coordinate axis $j_{0,0}$.

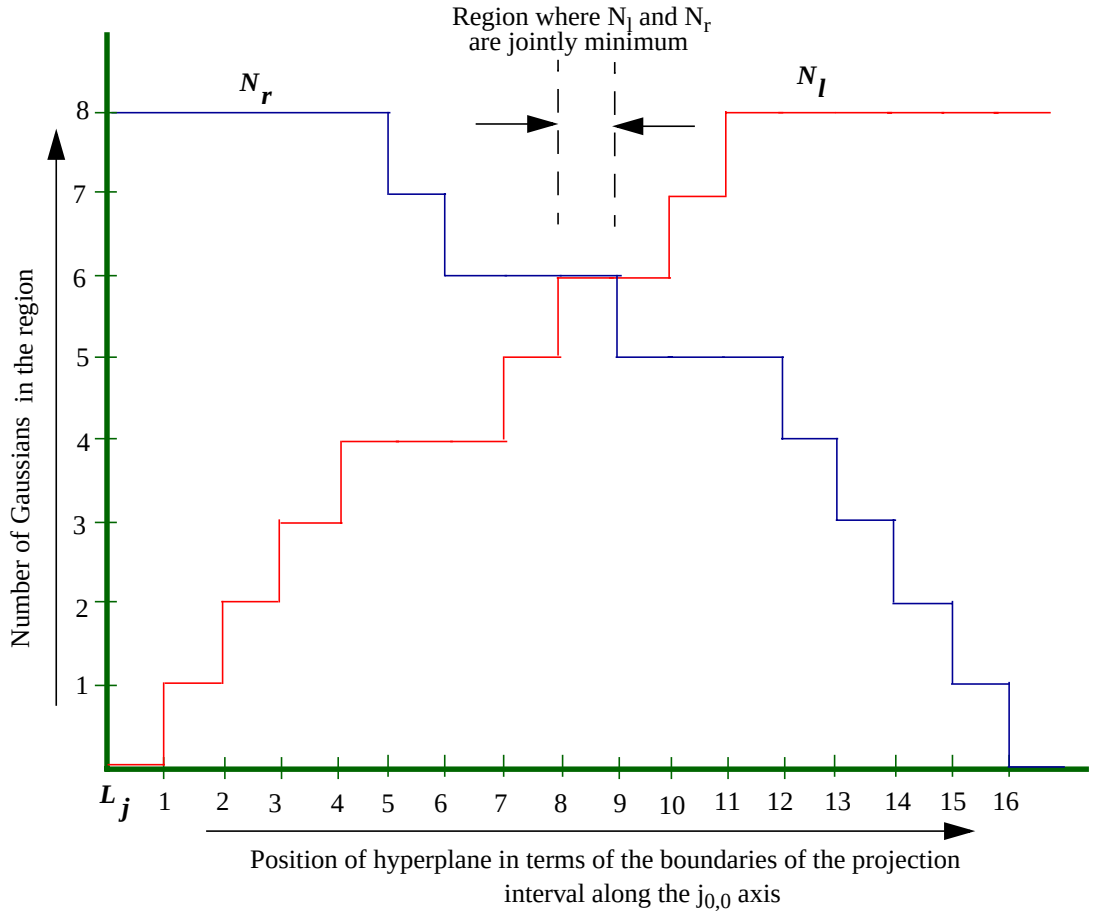

Figure 8. Behavior of the variables N_l and N_r with respect to the hyperplane $h_{0,0}$. Note that the projection boundaries of the Gaussians are shown to be evenly distributed along the horizontal axis in this figure.

a balanced division along j^{th} axis. Similarly, the balanced division hyperplane can also be found along the k^{th} coordinate axis.

For each coordinate axis in a k-dimensional region, there exists a projection boundary that results in a balanced division. Let N_{opt_i}, $i = 1, 2, \ldots, k$, be the number of Gaussians having non-empty intersections with the left and right regions, obtained as a

result of a balanced division along the j^{th} axis. The optimal hyperplane will be the one which results in the least value of N_{opt_i} over all coordinate axis [65].

Thus, the optimization process can be summarized as follows [59, 60]:

- For a bounded region R_b in a k-dimensional space, the balanced division is chosen for each coordinate axis;
- From these balanced divisions, the partition, for which corresponding (N_{opt_i}, N_{opt_i}) point is closest to the $(N_{R_b}/2, N_{R_b}/2)$ point, is chosen. The proximity measure is a Euclidean distance. In other words, the final optimal partition is chosen as the one which results in the least number of splits.

The balanced hyperplane for a coordinate axis in a k-dimensional region can be obtained as follows:

- Label the projection boundaries of the Gaussians in the region with L and U for the lower and upper projection boundaries respectively along the coordinate axis;
- Place the hyperplane at a position so that the number of lower projection boundaries (L) in the left region is equal to the number of upper projection boundaries (U) in the right region.

2.4. Approximation Error Introduced by the K-D Tree

In a k-d tree framework, the evaluation of a Gaussian is restricted to a Gaussian box with threshold T. This introduces an approximation error E_A in the computation of the observation probability [39, 59, 60]. The approximation error is bounded by the threshold T, such that $E_A \leq T$. In the BBI algorithm, the Gaussian boxes with threshold T are used. Also, only those Gaussians are used in the computation of observation probabilities whose boxes contain the feature vector. If M is the number of Gaussians in

an HMM state and w_m, $m = 1, 2, \ldots, M$, is the mixture coefficient for m^{th} Gaussian, then the mixture coefficients satisfy the constraints $w_m \geq 0$ and $\sum_{m=1}^{M} w_m = 1$. The approximation error introduced by each Gaussian is E_A. Thus, the overall approximation error can be written as $\sum_{m=1}^{M} w_m \cdot E_A$ [54]. Using the constraints mentioned above, the overall approximation error is E_A. Since $E_A \leq T$, the overall approximation error is less than or equal to the threshold T. The Gaussian speedup and approximation error depend on tree parameters, namely error threshold T and tree depth d

We have discussed the BBI algorithm in this chapter. In the next chapter, we will discuss the issues related to the previous implementations of the BBI algorithm and suggest solutions for these issues. These modifications will lead to the Extended Bucket Box Intersection algorithm proposed in this thesis.

CHAPTER 3

EXTENDED BUCKET BOX INTERSECTION ALGORITHM

This chapter demonstrates various drawbacks of the BBI algorithm. The Extended BBI algorithm, proposed in this thesis, gives solutions for many of these problems. The performance of the two algorithms are compared and discussed in detail.

3.1. A Modified Optimization Criterion for Multiple Minima

The BBI algorithm requires the selection of a balanced hyperplane that produces a minimum number of Gaussian splits [60]. In practical applications there may be more than one collection of splits which produce the same (minimum) number of Gaussians. These hyperplanes may yield the same speedup but may not yield an optimal approximation error. The BBI algorithm doesn't consider this scenario. The Extended BBI algorithm proposes a modified optimization criterion to allow for multiple minima. In this algorithm, the following steps are used to obtain the optimal division hyperplane:

- Find the variance of those coordinate axes whose hyperplanes produce a minimum number of Gaussian splits;
- Choose the hyperplane corresponding to the coordinate axis with the highest variance.

To explain this, let us consider the example shown in Figure 9. In this example, the coordinate axis j has a higher variance. Thus, according to the Extended BBI algorithm,

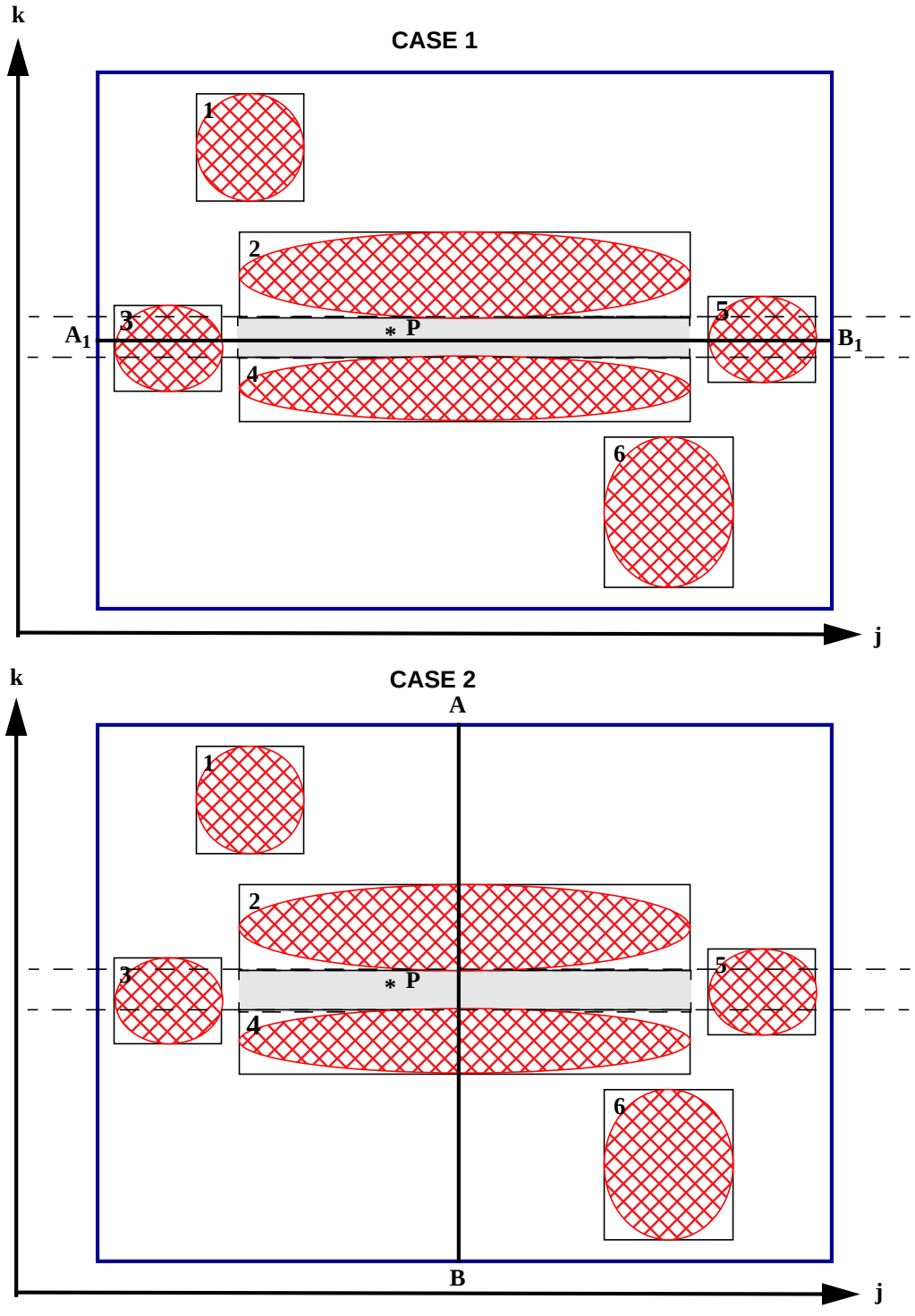

Figure 9. An example of multiple minima in the number of Gaussian splits. Case 1 contains a hyperplane A_1B_1 orthogonal to the axis with a lower variance and case 2 contains a hyperplane AB orthogonal to the axis with a higher variance.

the hyperplane orthogonal to this coordinate axis will yield an optimal result in the case of multiple minima. Using the optimization criterion proposed by the BBI algorithm, two balanced hyperplanes AB and A_1B_1, orthogonal to the coordinate axis j and k respectively, are found. Both the hyperplanes result in the same number of Gaussian splits, $N_{split} = 2$. The hyperplane A_1B_1 splits the region in such a way that the boxes corresponding to the Gaussians numbered 2 and 4 do not lie in same node of the tree.

Let us consider a feature vector P lying in the region shown in Figure 9. Given the positions and forms of the Gaussians in the region, Gaussians 2 and 4 can contain the vector P with equal probability. Under the framework of the division hyperplane A_1B_1, the vector P lies in Gaussian no. 2 only, which is contradictory to the fact that it lies in both the Gaussian boxes with equal probability. The k-d tree obtained using this division hyperplane will produce a higher approximation error.

Now consider the hyperplane AB. It splits the region into two balanced regions such that the nodes associated with both the regions contain Gaussians 2 and 4. Under this hyperplane, the feature vector P lies in the left region and Gaussians 2 and 4 both contain the vector. Thus, the hyperplane AB produces an optimal division for the vector P. The k-d tree obtained using this hyperplane will produce a lower approximation error. Generally speaking, all the vectors lying in the shaded region between Gaussians 2 and 4 lie in both the Gaussians with equal probability. For all these vectors, hyperplane AB produces a lower approximation error. Thus the hyperplane orthogonal to the axis with the highest variance is the optimal hyperplane.

Let us consider another example shown in Figure 10. The coordinate axis k has a higher variance than the axis j. In order to obtain a k-d tree, a balanced hyperplane is found for both coordinate axes. In case 1, the balanced hyperplane A is orthogonal to the axis having a lower variance. This hyperplane produces a division in which Gaussian 2 lies only in the right child of the node. Thus, this hyperplane yields an unbalanced distribution for Gaussian 2. In an unbalanced distribution, a Gaussian that has a high probability to be the most significant Gaussian for a feature vector in that region, lies in only one region. In this case there is a 50% probability that this Gaussian will not be used in approximate score computation. This is because the Gaussians lying in only one region

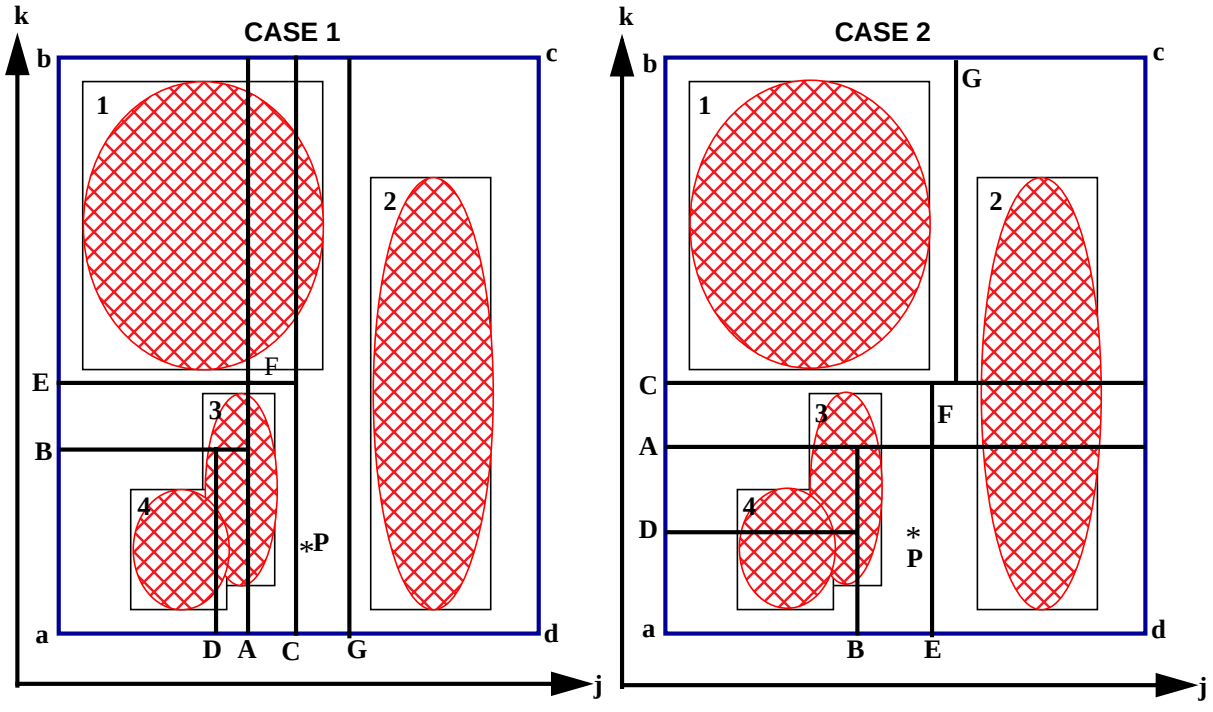

Figure 10. An example of multiple minima in the number of Gaussian split. Case 1 uses separating hyperplanes which are orthogonal to the axis with a lower variance while case 2 uses the hyperplane which is orthogonal to the axis with a higher variance.

are used for the score computation. Thus, a hyperplane, which gives an unbalanced distribution for the Gaussians in a region, gives a higher approximation error. In case 2, the balanced hyperplane A is orthogonal to the axis k which has a higher variance. We can observe that this hyperplane produces a balanced division for the Gaussian box 2 in both regions. This hyperplane will result in a lower approximation error.

According to the optimization criterion used in case 1, a 2-dimensional feature vector P lies in the bucket containing Gaussian box 1. Given the positions and forms of the Gaussians in the region, it can be observed that the Gaussian associated with this box can not be considered a significant Gaussian for the vector P. Thus, the optimization criterion proposed by the BBI algorithm will produce a larger approximation error. Under the division produced by the balanced hyperplanes used in case 2, the vector P lies in the Gaussian no. 3. This Gaussian can be considered as a significant Gaussian for the vector P since it has a minimum Euclidean distance from the vector P. Thus, the resulting k-d tree will produce a lower approximation error. The above discussion suggests that the optimization criterion proposed by the Extended BBI algorithm gives a lower WER.

To compare the performance of the optimization criterion used by the two algorithms, experiments were carried out using the OGI Alphadigits database [67]. We used 64-mixture cross-word triphone models. No fast Gaussian evaluation techniques were used in the baseline system. This system gave a 10.1% WER and used 57.9% of the total CPU time for the Gaussian Computations. Performance of the BBI and the Extended BBI algorithms is shown in Table 2. The performance of both algorithms is evaluated by

calculating the increase in WER of the system. The speedup obtained by both the algorithms is the same in this case. A k-d tree with a depth 6 and an error threshold 0.4 was used for both the algorithms.

Algorithm	Tree Depth (d)	Threshold (T)	WER (%)
Extended BBI	6	0.4	10.4
BBI	6	0.4	11.4

Table 2. The performance of the Alphadigits system using the BBI and Extended BBI algorithms.

The Extended BBI algorithm gave a WER of 10.4%. On the other hand, the BBI algorithm gave a WER of 11.4%. Thus, the Extended BBI algorithm introduced a significantly lower approximation error as compared to the BBI algorithm. In this example, the optimization criterion used by the Extended BBI algorithm gave an overall improvement of 10% in WER with the same amount of speedup in Gaussian evaluations. This improvement is statistically significant according to the Matched Pairs Sentence-Segment Word Error (MAPSSWE) test from NIST [80].

3.2. Mixture Weight Re-Normalization

The BBI algorithm finds the most significant Gaussians [54] for the evaluation of the mixture and uses only those Gaussians for the score computation. In this case it is required to re-normalize the mixture weights. But the BBI algorithm doesn't re-normalize the mixture weights [39, 54]. Further discussion will explain the re-normalization technique and its effects on the performance of the algorithm.

Let us consider that w_m, $m = 1, 2, \ldots, l, \ldots, M$, is the weight of the m^{th} Gaussian in a mixture of M Gaussians. Let us also consider that a feature vector $\bar{x}$ lies in a bucket

containing Gaussians l and $l+p$. We restrict the mixture evaluation of the vector $\bar{x}$ to Gaussians l and $l+p$ only. For re-normalization of mixture weights, we modify the weights to w_l^N and w_{l+p}^N, such that, $w_l^N = w_l/(w_l + w_{l+p})$ and $w_{l+p}^N = w_{l+p}/(w_l + w_{l+p})$. The modified mixture weights satisfy the constraint $w_l^N + w_{l+p}^N = 1$. Let us consider that the multivariate Gaussian has a mean vector $\mu = [\bar{\mu}_1, \bar{\mu}_2, \ldots, \bar{\mu}_M]$ and a diagonal variance $\Sigma = [\bar{\sigma}_1^2, \bar{\sigma}_2^2, \ldots, \bar{\sigma}_M^2]$, where $\bar{\mu}_m$ is a k-dimensional mean vector representing the mean of the m^{th} Gaussian and $\bar{\sigma}_m^2$ is a k-dimensional vector representing the diagonal covariance matrix of the m^{th} Gaussian. The log-likelihood of a feature vector $\bar{x}$ in such a k-dimensional space can be written as,

$$L = \sum_{m=1}^{M} \log(w_m \cdot P_m), \tag{11}$$

where,

$$P_m = \frac{1}{\sqrt{(2\pi)^k \prod_{i=1}^{k} \sigma_{im}^2}} \cdot \exp\left(-\frac{1}{2}\sum_{i=1}^{k}\left(\frac{x_{im} - \mu_{im}}{\sigma_{im}}\right)^2\right). \tag{12}$$

Considering the example discussed earlier, the log-likelihood of the feature vector using the BBI algorithm can be written as,

$$L = \log(w_l \cdot P_l) + \log(w_{l+p} \cdot P_{l+p}). \tag{13}$$

Using re-normalization of mixture weights, the log-likelihood can be written as,

$$L = \log\left(\frac{w_l}{w_l + w_{l+p}} \cdot P_l\right) + \log\left(\frac{w_{l+p}}{w_l + w_{l+p}} \cdot P_{l+p}\right). \quad (14)$$

If $w_l + w_{l+p} = C$ then the Equation 14 can be written as,

$$L = \log\left(\frac{w_l}{C} \cdot P_l\right) + \log\left(\frac{w_{l+p}}{C} \cdot P_{l+p}\right), \quad (15)$$

or,

$$L = \log(w_l \cdot P_l) + \log(w_{l+p} \cdot P_{l+p}) - 2 \cdot \log C. \quad (16)$$

As $C \leq 1$, $\log C \leq 0$. Thus, the log-likelihood using the normalized weights is obtained by subtracting a number that is less than or equal to zero, which is the same as adding a number which is greater than or equal to zero. This suggests that the use of re-normalized weights results in higher log-likelihoods, which should in turn lead to an improved system performance. Such a system produces the same amount of speedup as that obtained by a system using non-normalized mixture weights. The Extended BBI algorithm uses re-normalized mixture weights for the score computations.

To verify the effect of re-normalization, experiments were run on the OGI Alphadigits database using 64-mixture cross-word triphone models. Performance for the BBI and the Extended BBI algorithms is shown in Table 3. The baseline system gave a WER of 10.1%. The BBI algorithm gave a WER of 11.4%. The Extended BBI algorithm gave a WER of 11.1%. Thus, the BBI algorithm introduced a 13% relative approximation error whereas the Extended BBI algorithm introduced only a 10% relative increase in the WER. Thus, the approximation error introduced by the Extended BBI algorithm is smaller than that introduced by the BBI algorithm.

Algorithm	Tree Depth (d)	Threshold (T)	WER (%)
Extended BBI	6	0.4	11.1
BBI	6	0.4	11.4

Table 3. The performance of the Alphadigits system using the BBI and Extended BBI algorithms, where the Extended BBI algorithm is using re-normalized mixture weights.

The improvement gained by the mixture weight re-normalization is not statistically significant according to the MAPSSWE test. This is because the mixture weight re-normalization results in a small increase in the log-likelihood. The average increase in log-likelihood is given as,

$$\Delta L = -\left(m \cdot \log\left\{ \sum_{m=1}^{M} w_m \right\} \right)_{m \in bucket} . \quad (17)$$

If the parameters of the BBI algorithm are selected such that the algorithm restricts the score computation of a vector to only 25% of Gaussians representing the state, the sum of the mixture weights used for score computations will approximately be equal to 0.25[1]. Thus the average increase in log-likelihood will be $\Delta L \approx -4.\log(0.25) = 5.56$. Since, the increase in the log-likelihood is small, the resulted improvement in WER of the system using re-normalized weights is small.

1.The value of ΔL will be 0.25 in this case, if all the Gaussians representing the state have same mixture weight.

3.3. Selection of a K-D Tree

There are several options for how we apply the k-d tree to the HMM model: one k-d tree per state, one k-d tree shared by all states, or multiple k-d trees with each tree sharing a subset of the states. The optimal choice depends on the task, available memory and other factors. Previous implementations of the BBI algorithm have successfully employed the latter two approaches [39, 54]. Use of one k-d tree for all the states yields a very high speedup and requires small amounts of memory. However, the main disadvantage is that it results in a very large approximation error. Similarly, when multiple trees are used, the speedup in Gaussian evaluations can be increased by using tree-tying and similar techniques [13, 15]. But such implementations also result in a large approximation error. The Extended BBI algorithm presented here uses one k-d tree per HMM state. The use of one k-d tree per state results in a smaller reduction in CPU time. However, it delivers a smaller approximation error. Thus, it results in a lower WER.

In order to compare the performance of the system under these k-d tree choices, the experiments were run using the OGI Alphadigits database with 64-mixture cross-word triphone models. The WER of the two systems is given in Table 4. The BBI algorithm gave a WER of 13.6% whereas the Extended BBI algorithm gave a WER of 10.9%. Thus, the BBI algorithm introduced a 35% relative increase in WER whereas the Extended BBI algorithm introduced only an 8% increase. However, for the Extended BBI algorithm, Gaussian evaluations used 29.1% of the total CPU time, while for the BBI algorithm, Gaussian evaluations used only 24.7% of the total CPU time. Hence, the Extended BBI

Algorithm	Tree Depth (d)	Threshold (T)	% of Total CPU Time used by Gaussian Evaluations	WER (%)
Extended BBI	6	0.4	29.1	10.9
BBI	6	0.4	24.7	13.6

Table 4. The performance of the Alphadigits system using the BBI and Extended BBI algorithms. The BBI system is using a single large tree shared by all states while the Extended BBI system is using one k-d tree per state.

algorithm is a slightly less efficient. The reduction in WER obtained by the Extended BBI algorithm is statistically significant according to the MAPSSWE test.

Thus far we have studied the improvements in WER of a system utilizing the Extended BBI algorithm with a modified optimization criterion, mixture weight re-normalization, and state-level k-d trees. Next, we evaluated the performance of the Extended BBI algorithm using these algorithms in combination. The same experimental setup was used for this experiment. Performance for the BBI and the Extended BBI algorithms is shown in Table 5. The BBI algorithm gave a 11.4% WER whereas the Extended BBI algorithm gave a 10.2% WER. The Extended BBI algorithm gave a 12% relative improvement over the BBI algorithm. This improvement is statistically significant according to the MAPSSWE test.

Algorithm	Tree Depth (d)	Threshold (T)	WER (%)
Extended BBI	6	0.4	10.2
BBI	6	0.4	11.4

Table 5. The performance of the BBI and Extended BBI algorithms, where the Extended BBI algorithm is using state level k-d trees, a modified optimization criterion and mixture weight re-normalization.

In this chapter, we have compared the performance of the Extended BBI algorithm and the BBI algorithm on a limited task to calibrate the impact of our proposed enhancements. In the next chapter, we will evaluate the performance of the Extended BBI algorithm across a wide variety of experimental conditions.

CHAPTER 4

EXPERIMENTS

The last three chapters provided a theoretical background for the Extended BBI algorithm. In this chapter, we present the performance of the proposed algorithm on various industry-standard databases.

4.1. Experimental Databases

This section discusses the databases that have been used in this thesis to measure the performance of the Extended BBI algorithm.

TIDIGITS

We have used TIDIGITS database for the initial evaluation of the algorithm [32]. TIDIGITS is a small database containing continuous digits. The vocabulary size of the database is 11 words, and contains the digits “0-9” and “oh”. Many state-of-the-art systems produce a WER of 0.2% on the TIDIGITS task [39].

The experiments for this task used 16-mixture components. These experiments used 5-state left-to-right models containing a start and a stop state [23]. The input features used FFT-derived MFCC (Mel-frequency Cepstrum Coefficients) [25]. Without using the

Extended BBI algorithm, the system produced a WER of 0.6%. Also, Gaussian evaluations took 24.4% of the total CPU time in this system.

Alphadigits

The OGI Alphadigits corpus is a database of digits sequences that are six digit strings of letters and numbers [67, 68]. It has a vocabulary of 36 words [69]. The experiments used standard 39-dimensional MFCC feature vectors. To compare the performance of the algorithm as a function of the number of mixture Gaussians, 16, 32 and 64-mixture components were used.

Without using the Extended BBI algorithm, the system produced a WER of 10.3% for 16-mixture and 32-mixture components and a WER of 10.1% for 64-mixture components. In this system, Gaussian evaluations took 31.7% of the total CPU time for 16-mixture, 43.2% of the total CPU time for 32-mixture and 57.9% of the total CPU time for 64-mixture components.

SWITCHBOARD

SWITCHBOARD (SWB) is a large vocabulary database, which is commonly used to evaluate the performance of the LVCSR systems [70]. The database was collected by Texas Instruments (TI) in 1990. It contains over 2,000 two-sided conversations. The database presents many challenges to the LVCSR systems [71, 72].

The experiments for this task used 12-mixture components. The input feature vectors used standard FFT-derived cepstral coefficients with cepstral mean subtraction. The decoding performed an acoustic rescoring of input lattices. The language model

scores were taken from the input lattices. Without using the Extended BBI algorithm, the system produced a WER of 41.1%. In this system, Gaussian evaluations took only 9.9% of the total CPU time.

4.2. Measurement of Performance

The performance of the algorithm is measured by the speedup in mixture component evaluations and the approximation error. The speedup in Gaussian evaluation is computed as follows:

$$\%\ Gaussian\ Speedup\ =\ \frac{(T(CPU)_{Baseline} - T(CPU)_{FGC})}{T(CPU)_{Baseline}} \times 100 \qquad (18)$$

where,

- $T(CPU)_{Baseline}$ - percentage of the total CPU time used by the Gaussian evaluations in the baseline system,
- $T(CPU)_{FGC}$ - percentage of the total CPU time used by the Gaussian evaluations in the system using fast Gaussian algorithm.

4.3. Profiling Tool

To compute the percentage of CPU time used by Gaussian evaluations, we have used the gprof [73] utility. Appendix A provides the motivation to use this utility and also establishes its accuracy. The gprof utility produces a dynamic call graph and a profile file (gmon.out by default) for a program. Using the call graph and the profile file, a flat profile is obtained that gives the statistics containing the total execution time (as a percentage of total time) of each function and the number of times the function is called during the complete execution of the program.

4.4. Effects of Algorithm Parameters on the Algorithm Performance

The Extended BBI algorithm has two parameters — the k-d tree depth and an error threshold. The Gaussian speedup and the approximation error depend on both the parameters.

Effect of Tree Depth on Speedup and Approximation Error

An increase in the tree depth increases the number of separating hyperplanes and the number of buckets in the leaf node of the tree [54]. As the number of buckets increases, a k-d tree with a lower number of Gaussians in each bucket may be created [39]. This results in a reduced number of significant Gaussians for the Gaussian evaluations. Thus, a higher tree depth produces a higher Gaussian speedup. Also, a reduced number of Gaussians for score computations produces a higher approximation error [54]. Therefore an increase in the tree depth increases the approximation error.

Effect of Threshold on Speedup and Approximation Error

The Gaussian threshold defines the size of the Gaussian boxes used to build the k-d tree. As the threshold increases, the size of the Gaussian boxes decreases. This may produce a reduced overlap between Gaussian boxes. This gives a k-d tree with a relatively smaller number of Gaussians in each bucket of the tree and produces a higher speedup. This also reduces the computational load of the Gaussian evaluations. Thus, as the number of Gaussians used for the score computations decreases, the approximation error introduced by the algorithm increases [39].

The error threshold for the Gaussian boxes is chosen such that it is smaller than the maximum value of the Gaussians. This is known as an “absolute” threshold because it is an absolute value with respect to the maxima of the Gaussian. In practical ASR systems, the mixture of Gaussians contains Gaussian mixture components with different maxima. In some systems, the maxima may differ by more than an order of magnitude. Gaussians with lower mixture weights represent lower probability regions of the mixture. If the error threshold is chosen such that it is smaller than the maxima of the Gaussian with the highest probability, but greater than the maxima of the Gaussians with lower probabilities, it will result in an inaccurate modeling of the Gaussians with lower probabilities. To avoid this problem we use a *relative* threshold.

4.5. Relative Threshold

A threshold chosen based on a certain percentage of each maximum for each Gaussian mixture component is known as a relative threshold [54]. A relative threshold of 0.5 indicates that each Gaussian is cut at 50% of its maximum value. In this case, even the Gaussians with very smaller maxima can participate in k-d tree generation. This results in a lower approximation error. Thus, the performance of the algorithm using a relative threshold is better than that obtained using an absolute threshold.

The relative threshold, T^R, is bounded, such that, $0 \leq T^R \leq 1$. In the case of a relative threshold, only the exponential part of the Gaussian equation will contribute to the projection boundary, and we can write:

$$T^R = \exp\left(-\frac{1}{2}\sum_{i=1}^{k}\frac{(x_i-\mu_i)^2}{\sigma_i^2}\right). \quad (19)$$

Solving Equation 19 for the coordinate axis i gives the projection boundaries of the Gaussian box along this axis as follows:

$$x_i = \mu_i \pm \sqrt{-2\cdot\sigma_i^2\cdot\log(T^R)}\ . \quad (20)$$

4.6. Experimental Results and Analysis

The parameters of the Extended BBI algorithm should be chosen to achieve a significant speedup with minimal approximation error. This is done by analyzing the performance of the algorithm using various combinations of parameters and finding the parameter pair which gives best results. A publicly available speech-to-text system [74] has been used in this thesis to evaluate the performance of the proposed algorithm.

TIDIGITS

TIDIGITS [32] was used for the initial evaluation of the Extended BBI algorithm. The relative threshold used by the algorithm was varied from 0.2 to 0.8 in steps of 0.1. K-d trees with depth between 3 and 5 were used. Table 6 gives the performance of the system which is not using the Extended BBI algorithm (No EBBI System) and the performance of the system which is using the Extended BBI algorithm.

We can observe that for a threshold of 0.2, the Gaussian speedup increased from 32.4% to 41.2% as the tree depth increased from 3 to 5. For these values of parameters, the WER of the recognition system remained at 0.6%. This suggests that the proposed

Tree Depth	Relative Threshold	% of Total CPU Time Taken by Gaussian Score Evaluations	Gaussian Speedup (%)	WER (%)
No EBBI System		24.38	-	0.6
3	0.2	16.47	32.44	0.6
4	0.2	15.27	37.37	0.6
5	0.2	14.36	41.10	0.6
3	0.3	14.81	39.25	0.6
4	0.3	13.56	44.38	0.6
5	0.3	12.41	49.10	0.7
3	**0.4**	**13.41**	**45.00**	**0.6**
4	0.4	11.80	51.60	0.7
5	0.4	10.74	55.95	0.7
3	0.5	12.20	49.96	0.7
4	0.5	10.21	58.12	0.7
5	0.5	9.04	62.92	0.8
3	0.6	10.83	55.58	0.8
4	0.6	8.91	63.45	0.8
5	0.6	7.49	69.28	0.9
3	0.7	9.52	60.95	0.8
4	0.7	8.14	66.61	0.9
5	0.7	6.59	72.97	1.0
3	0.8	8.39	65.59	1.0
4	0.8	6.23	74.45	1.2
5	0.8	4.77	80.43	1.6

Table 6. Performance of the TIDIGITS system using the Extended BBI algorithm.

algorithm produced a 41.2% speedup without adding any additional approximation error. Further, it can be seen that a depth of 3 with a relative threshold of 0.4 produced a 45% speedup in score computations without degrading the performance of the system. Also, a

tree depth of 4 and a relative threshold of 0.5 produced a 58% speedup with only a 0.1% relative increase in the WER of the system. Thus, the algorithm produces a significant amount of speedup without increasing the WER of the system. Figure 11 gives the speedup in Gaussian evaluation as a function of relative threshold. The plots shown represent tree depths of 3, 4 and 5. From these results, we can observe that:

- For a given tree depth, the speedup in Gaussian evaluations increases with an increase in the relative threshold;
- For a given threshold, the speedup in mixture component evaluation increases with an increase in the k-d tree depth.

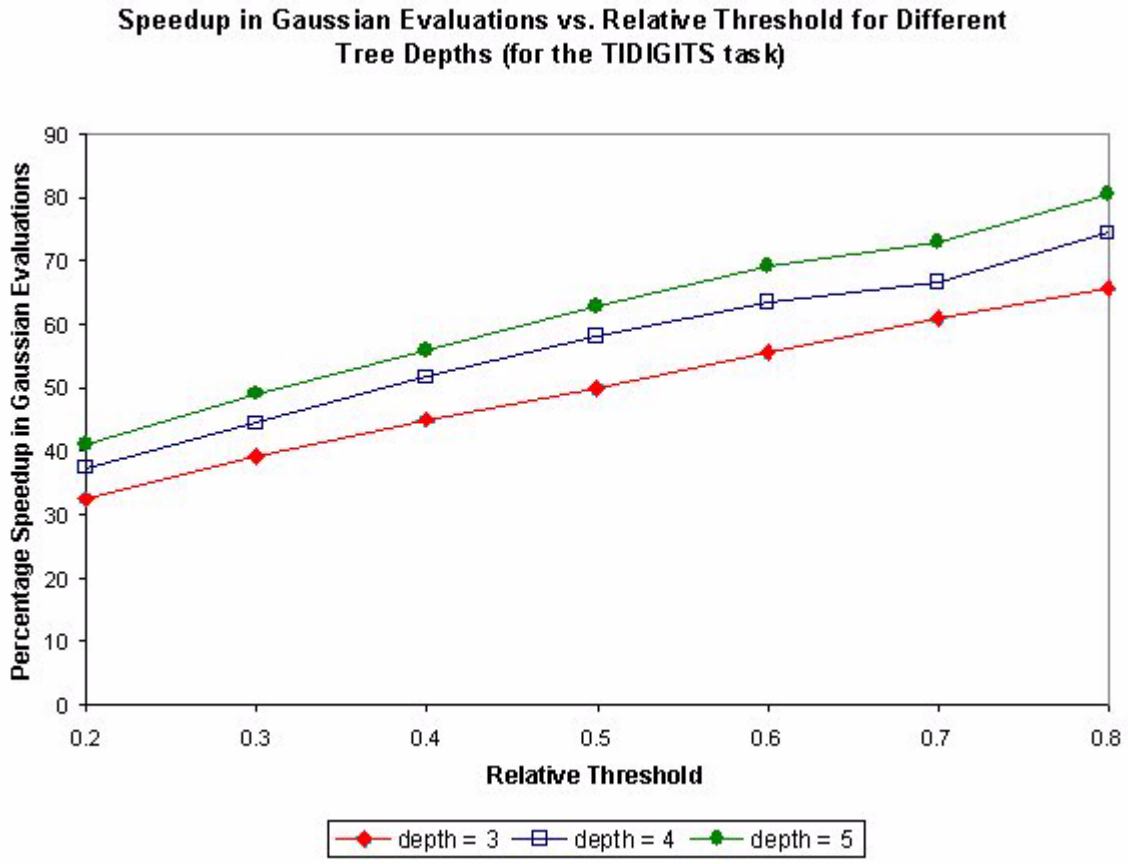

Figure 11. The effect of variations in relative threshold on the speedup in Gaussian evaluations for a TIDIGIT system with a constant k-d tree depth. K-d trees with depth 3, 4 and 5 were used.

Next, we will analyze the effect of the algorithm parameters on the WER of the system. The plots shown in Figure 12 give the variation in WER of the system as a function of threshold and tree depth. These results suggest that:

- For a constant tree depth, the WER increases with an increase in the relative threshold;
- For a constant relative threshold, the WER of the system increases with an increase in the tree depth.

It can be observed that the WER remained unchanged for lower values of tree depths and thresholds. Figure 13 gives a clear understanding of the algorithm

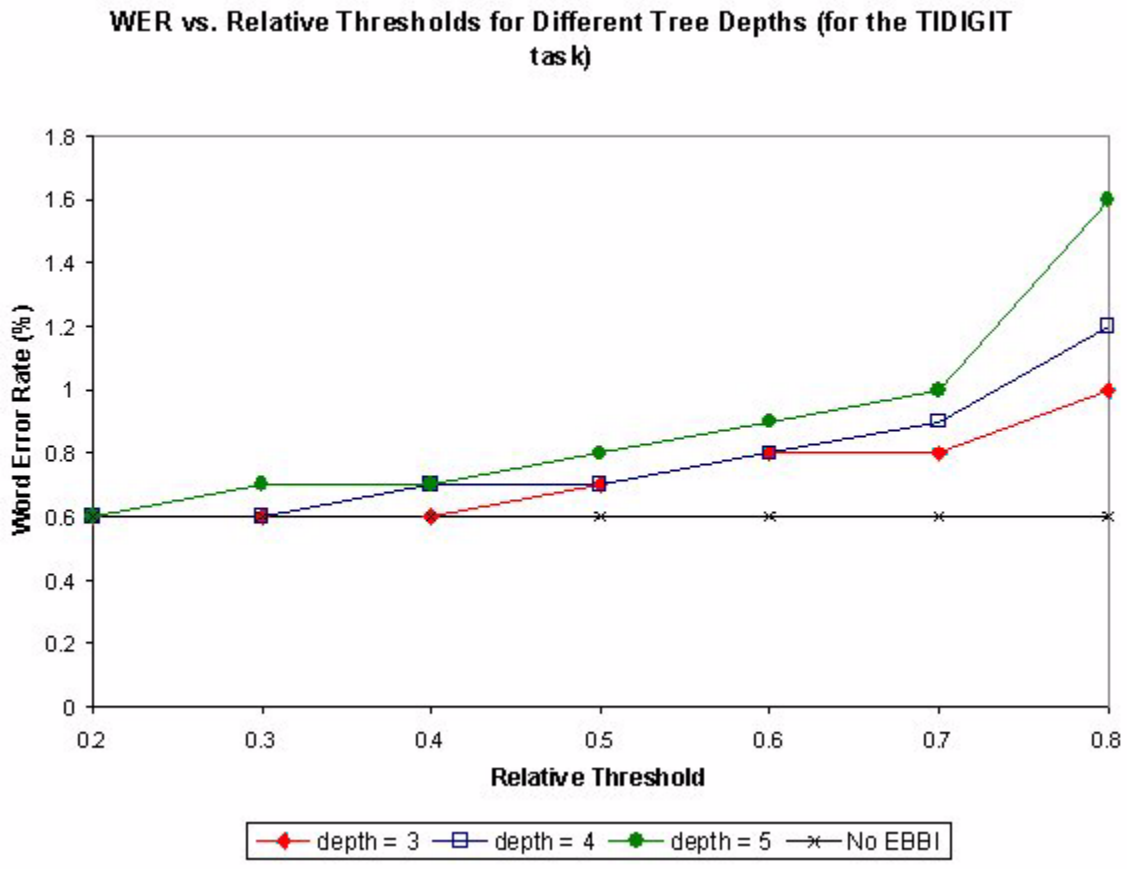

Figure 12. The effect of variations in the relative threshold on the WER of a TIDIGIT system with a constant k-d tree depth.

performance. This figure shows the Gaussian speedup and WER of the TIDIGITS system as a function of the algorithm parameters. The WER is shown along the vertical axis and the Gaussian speedup is plotted along the horizontal axis. The relative threshold was increased from 0.2 to 0.8 along the horizontal axis. Different colors in the plot correspond to different k-d tree depths. It can be observed that the maximum speedup that was obtained without any degradation in recognition performance was 45%. A higher speedup in Gaussian evaluations is obtained at the cost of a small increase in WER.

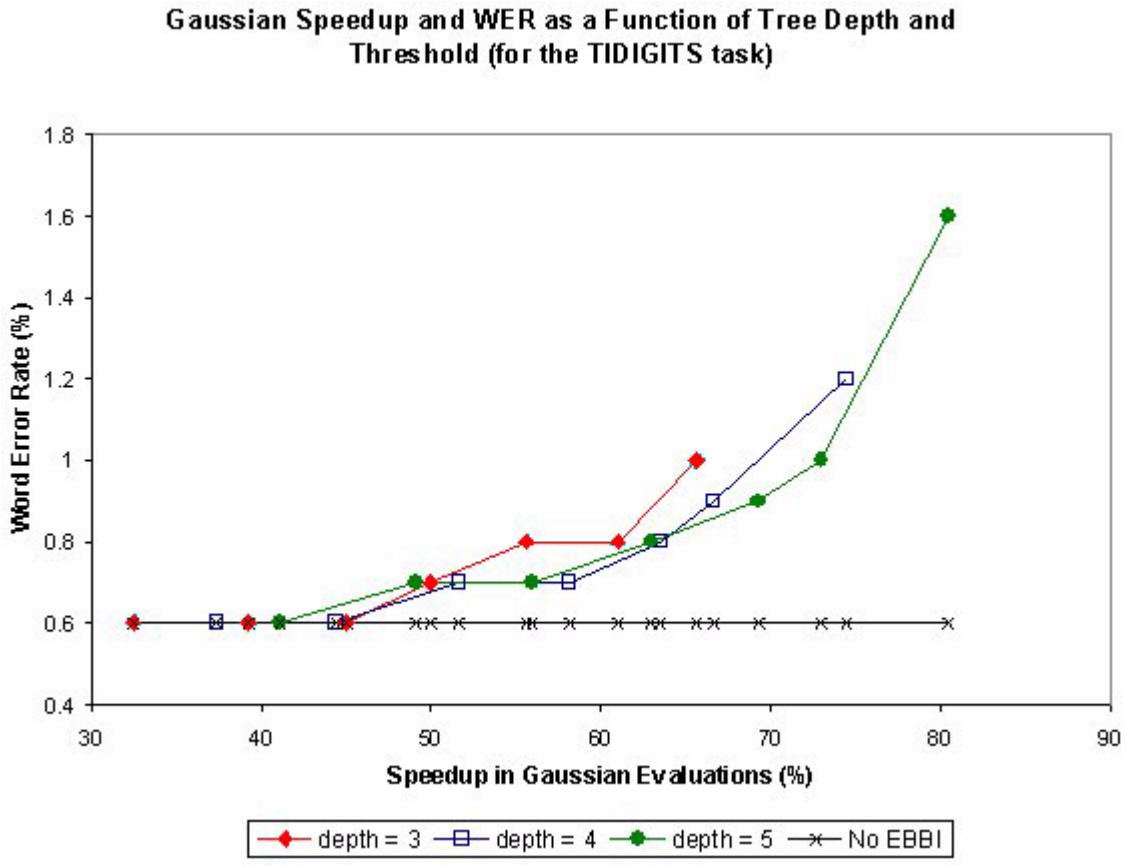

Figure 13. The effect of k-d tree depth and relative threshold on the WER and Gaussian speedup for the TIDIGITS system using the Extended BBI algorithm.

Alphadigits

The OGI Alphadigits task [75] was used to obtain the performance of the algorithm as a function of the number of mixture components. For this, 16, 32 and 64 mixture components were used. For the Alphadigits system with 16-mixture components, the relative threshold was varied from 0.2 to 0.8 at steps of 0.1. The k-d tree depth was varied from 3 to 5. Using the Extended BBI algorithm, the speedup in Gaussian evaluations and the WER of the system were obtained. The results are listed in Table 7. Gaussian evaluations took 31.7% of the total CPU time in the system, which is not using the Extended BBI algorithm. This system produced a WER of 10.3%. We can observe that for a constant threshold, the percentage speedup in Gaussian evaluations increases with an increase in the tree depth. But, this speedup is obtained at a cost of an increased WER. For a threshold of 0.2, as the tree depth increased from 3 to 5, Gaussian speedup increased from 31% to 39% without any increase in the WER of the system.

The variation in percentage speedup is shown as a function of algorithm parameters in Figure 14. The relative threshold is plotted along the horizontal axis and the Gaussian speedup is plotted along the vertical axis. The plots shown represent tree depths of 3, 4 and 5. It can be observed that for a fixed tree depth, Gaussian speedup increases with an increase in the relative threshold. For a depth of 5, as the threshold increased from 0.2 to 0.8, the speedup in Gaussian evaluations increased from 39% to 79%. For a threshold of 0.5, as the tree depth increased from 3 to 5, the Gaussian speedup increased from 46% to 58%.

Tree Depth	Relative Threshold	% of Total CPU Time Taken by Gaussian Score Evaluations	Gaussian Speedup (%)	WER (%)
No EBBI System		31.70	-	10.3
3	0.2	21.86	31.04	10.4
4	0.2	20.02	36.84	10.4
5	0.2	19.38	38.89	10.4
3	0.3	21.19	30.91	10.5
4	0.3	19.94	37.10	10.5
5	**0.3**	**18.68**	**41.07**	**10.5**
3	0.4	19.63	38.08	10.6
4	0.4	17.89	43.56	10.8
5	0.4	16.20	48.90	10.7
3	0.5	17.13	45.96	10.9
4	0.5	15.00	52.68	11.1
5	0.5	13.33	57.95	11.2
3	0.6	14.97	52.78	11.4
4	0.6	13.04	57.73	11.5
5	0.6	10.96	65.42	12.0
3	0.7	12.85	59.46	12.0
4	0.7	11.11	64.95	12.6
5	0.7	8.64	72.74	13.4
3	0.8	11.04	65.17	13.5
4	0.8	8.54	73.06	14.6
5	0.8	6.61	79.15	15.6

Table 7. Performance of the Alphadigits system using the Extended BBI algorithm. The system used 16-mixture components.

The variation in WER is shown as a function of the algorithm parameters in Figure 15. The plots indicate that for a fixed tree depth, the WER of the recognition system increases as the threshold increases. When the threshold is varied from 0.2 to 0.5,

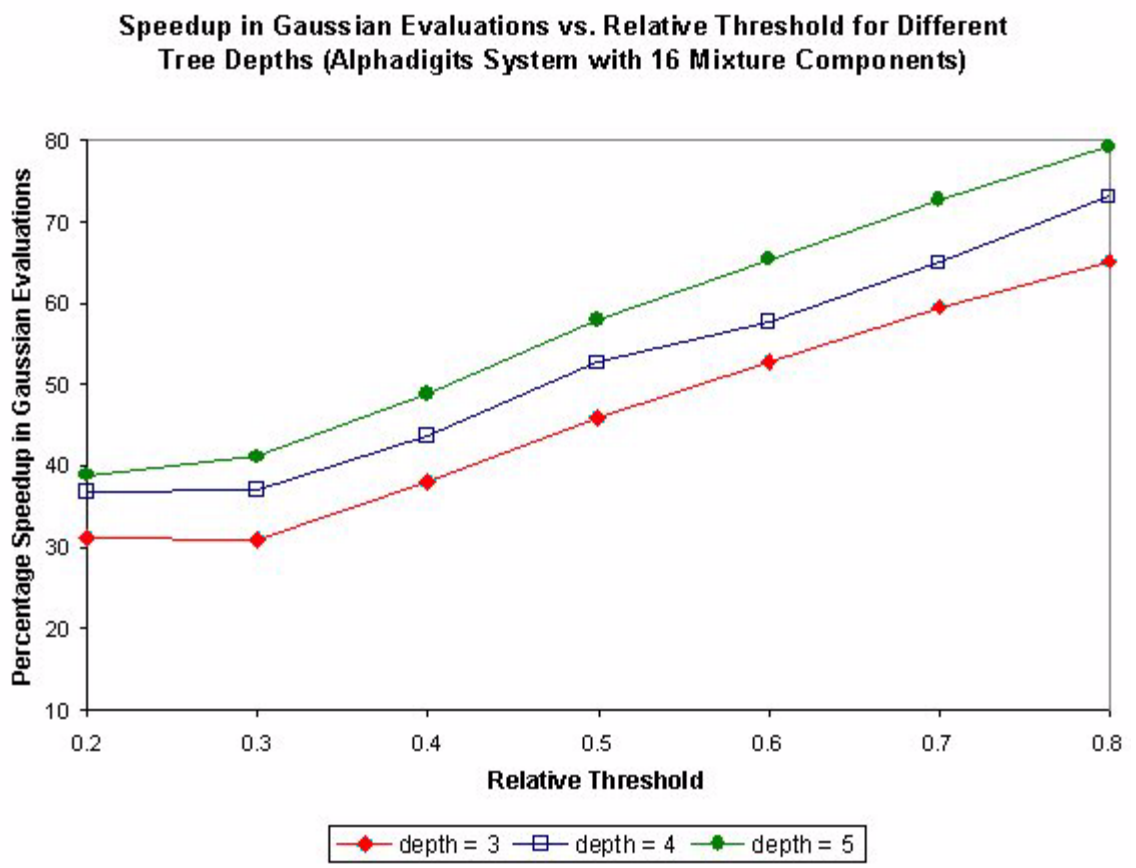

Figure 14. The variation in Gaussian speedup as a function of the relative threshold for an Alphadigits system with 16-mixture components. The plots correspond to the tree depths of 3, 4 and 5.

the variation in WER is small. But the variation in WER is significantly larger when the threshold is varied from 0.5 to 0.8. We can also observe that for a fixed value of the relative threshold, WER increases with an increase in tree depth. For a relative threshold of 0.7, when the tree depth increased from 3 to 5, the relative increase in WER increased from 16% to 30%.

Figure 16 gives the simultaneous effect of the algorithm parameters on the Gaussian speedup and the WER of the system. The relative threshold is varied from 0.2 to 0.8 along the horizontal axis. The tree depth is fixed for each plot. It can be observed that

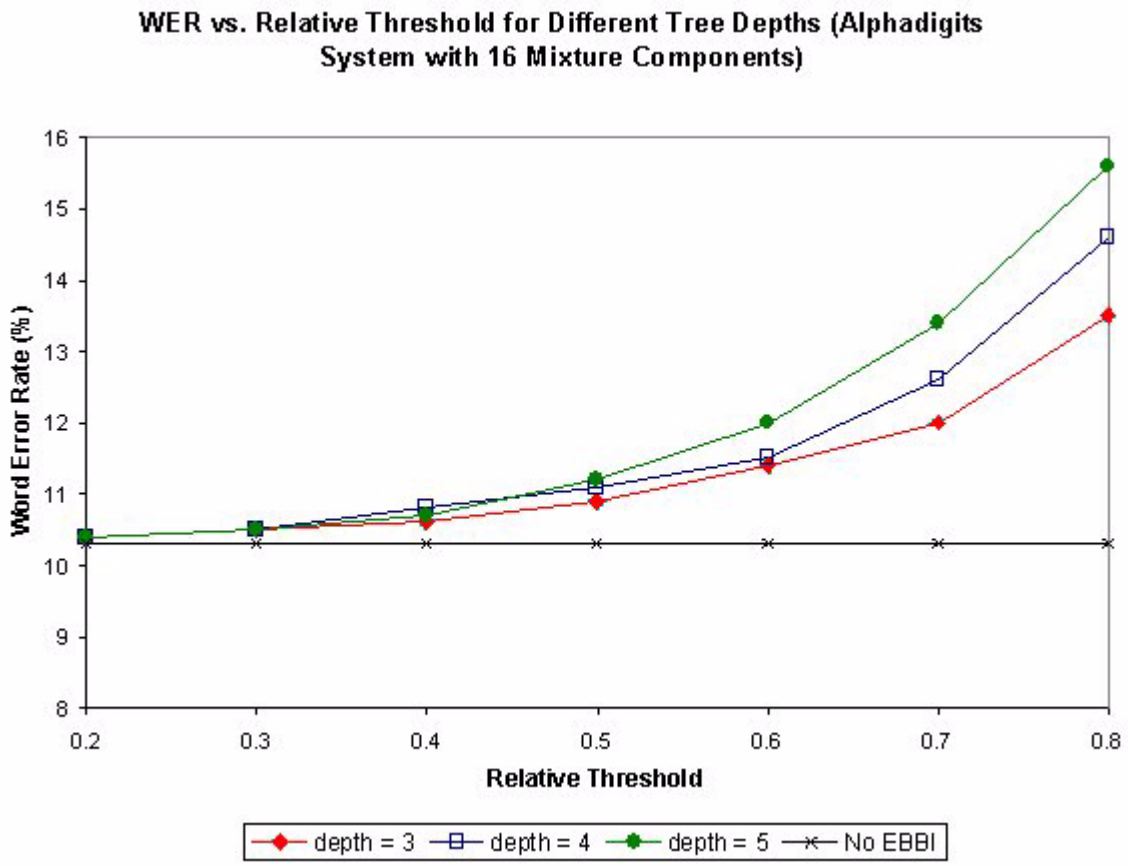

Figure 15. The effect of variations in relative threshold on the WER of an Alphadigits system with 16-mixture components for a constant tree depth.

the Gaussian speedup as well as the WER increases with an increase in threshold or tree depth. The Extended BBI algorithm with a tree depth of 5 and a threshold of 0.3 produced a 41% speedup with only a 2% relative degradation in WER. A higher speedup is obtained at the cost of a significantly higher increase in WER. Thus, a tree depth of 5 and a relative threshold of 0.3 can be considered as optimal values of the algorithm parameters for this task.

Next, we will consider the Alphadigits system [76] with 32-mixture components. The percentage of the total CPU time used for the evaluation of the mixture component

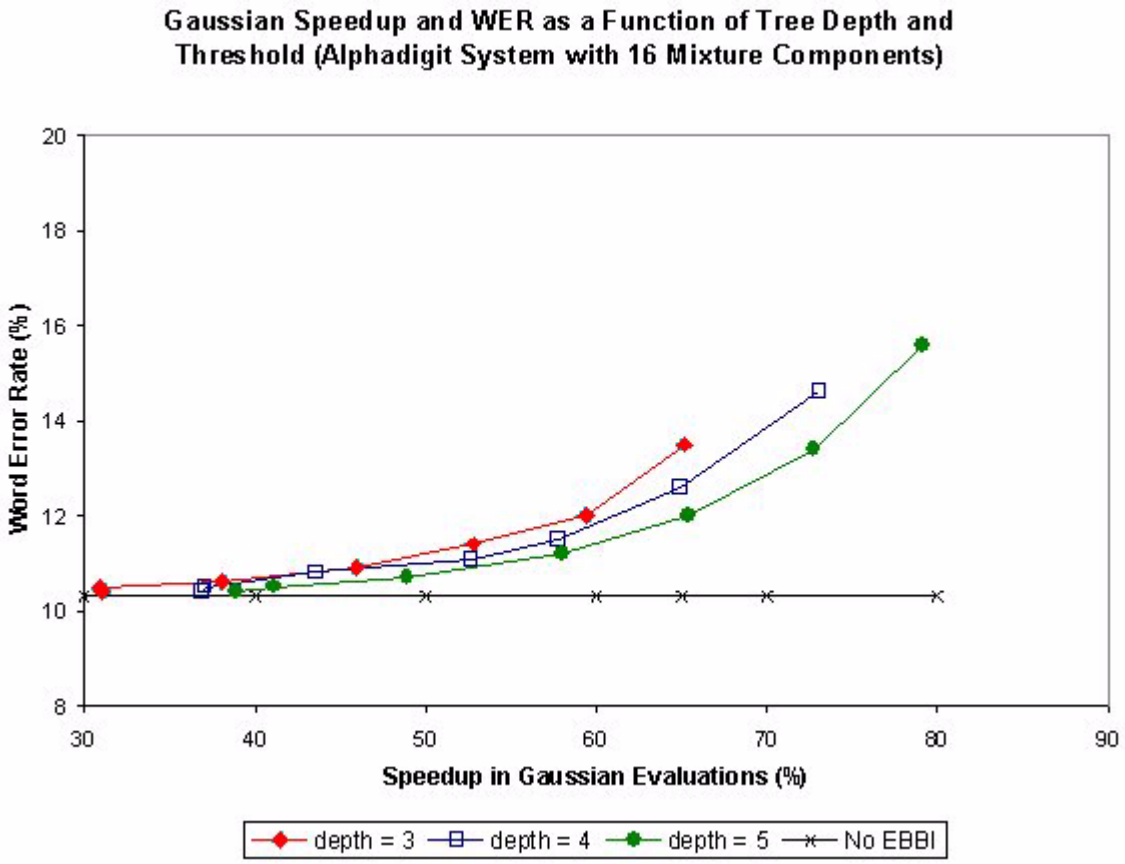

Figure 16. The effect of variations in tree depth and relative threshold on the WER and Gaussian speedup for an Alphadigits system using 16-mixture cross-word triphone models.

was 43% in this system, which is 36% higher than the CPU time used by the Gaussian evaluations in the system using 16-mixture components. K-d trees of depth 3, 4, 5 and 6 were used in the experiments. The relative threshold was varied from 0.3 to 0.8 for different test runs. The performance of the algorithm using different values of the algorithm parameters is shown in Table 8.

Figure 17 shows a variation in the Gaussian speedup as a function of the relative threshold. It can be observed that for a fixed tree depth, Gaussian speedup increases with an increase in the threshold. Also, for a fixed threshold value, a higher speedup is obtained

Tree Depth	Relative Threshold	% of Total CPU Time Taken by Gaussian Score Evaluations	Gaussian Speedup (%)	WER (%)
No EBBI System		43.17	-	10.3
3	0.3	29.99	30.53	10.3
4	0.3	26.65	38.26	10.3
5	0.3	24.41	43.46	10.3
6	**0.3**	**22.33**	**48.27**	**10.3**
3	0.4	26.92	37.64	10.4
4	0.4	23.99	44.43	10.4
5	0.4	21.64	49.87	10.5
6	0.4	19.03	55.92	10.6
3	0.5	24.43	43.41	10.6
4	0.5	21.76	49.59	10.7
5	0.5	19.04	55.90	10.8
6	0.5	16.45	61.89	10.9
3	0.6	22.39	48.13	10.9
4	0.6	20.12	53.40	11.2
5	0.6	16.99	60.64	11.4
6	0.6	14.90	65.49	11.7
3	0.7	21.70	49.73	11.4
4	0.7	17.93	58.47	11.7
5	0.7	14.77	65.79	12.2
6	0.7	11.37	73.66	12.8
3	0.8	19.05	55.87	12.4
4	0.8	14.60	66.18	13.0
5	0.8	10.89	74.77	14.2
6	0.8	7.98	81.51	15.1

Table 8. Percentage speedup in Gaussian evaluation and WER for an Alphadigits system with 32-mixture components.

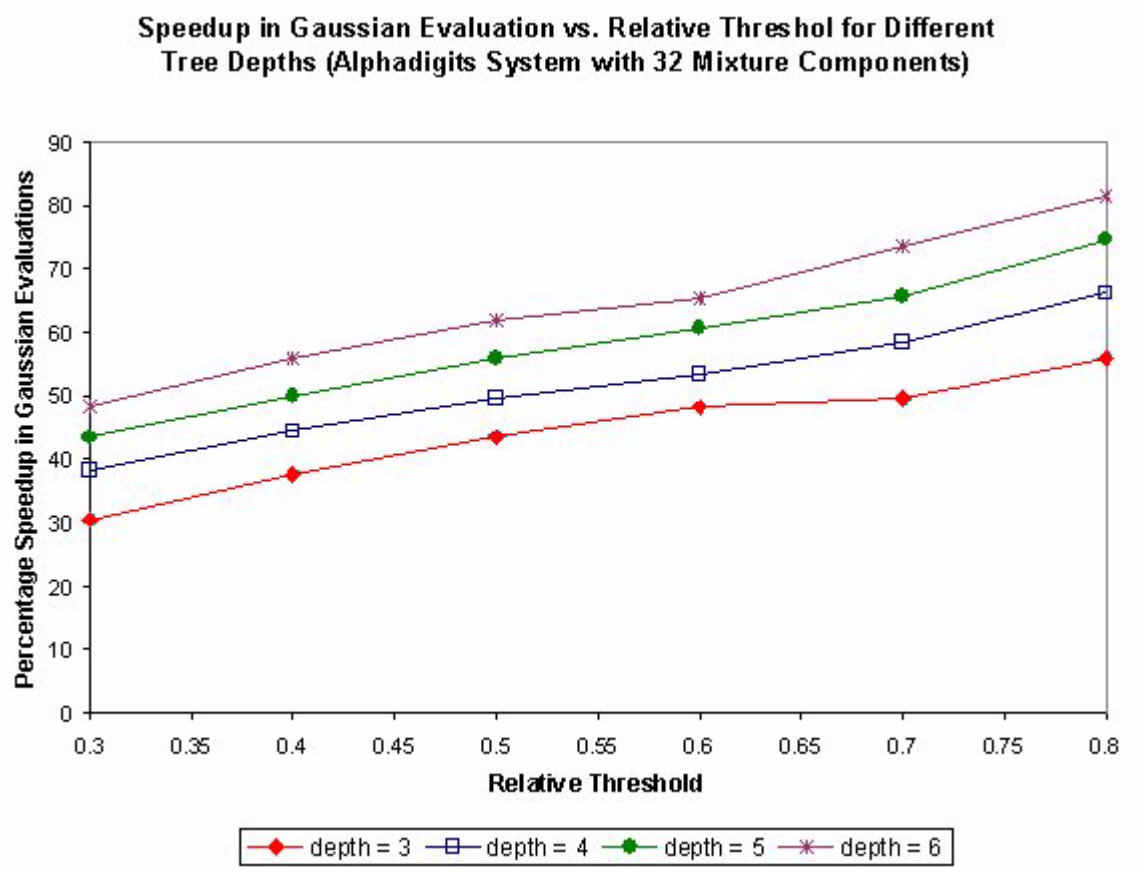

Figure 17. Variations in the Gaussian speedup as a function of the relative threshold and k-d tree depth for an Alphadigits system using 32-mixture cross-word triphone models.

as the tree depth increases. For a threshold of 0.3, as the tree depth varied from 3 to 6, the Gaussian speedup increased from 30% to 48%. Similarly, for a tree depth of 3, as the threshold increased form 0.3 to 0.8, the Gaussian speedup increased from 30% to 56%.

The variation in WER is shown as a function of algorithm parameters in Figure 18. It can be observed that the approximation error introduced by the Extended BBI algorithm increases as the tree depth or relative threshold increases. For a threshold of 0.4, as the tree depth increased from 3 to 6, the WER increased from 10.4% to 10.6%. In other words, the

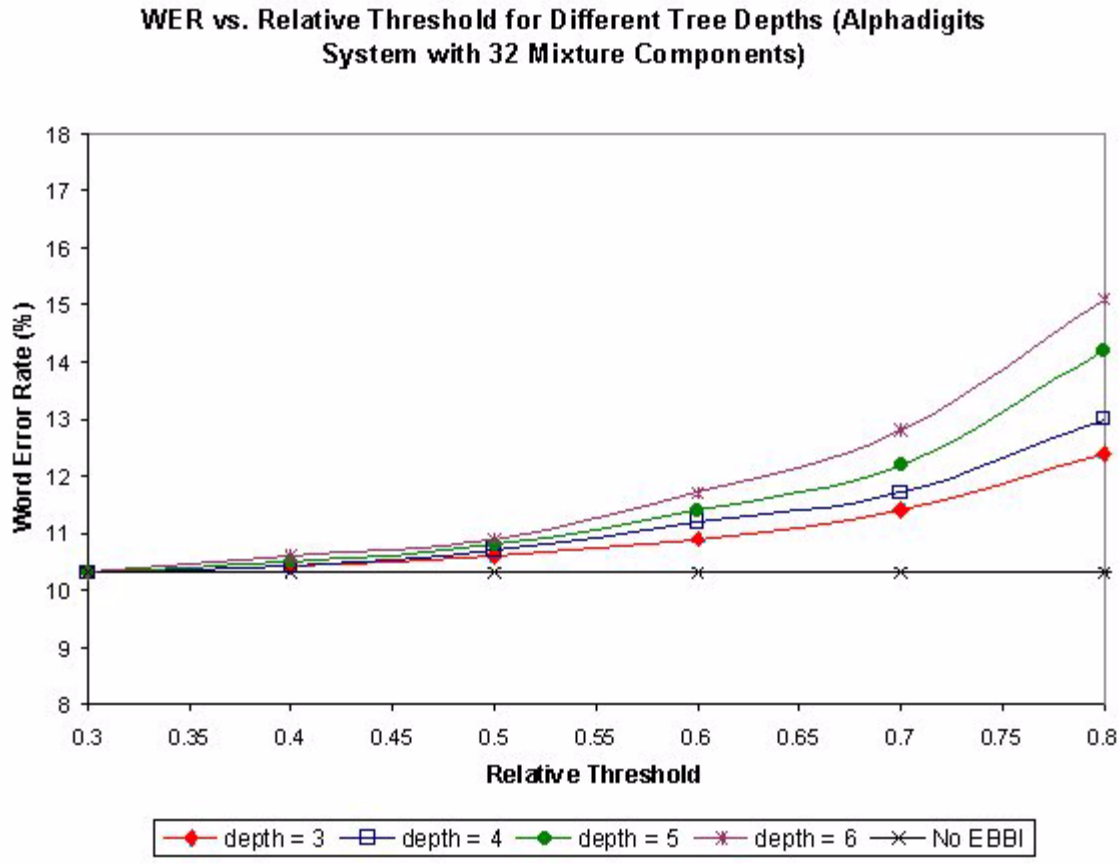

Figure 18. The effect of the algorithm parameters on the WER of an Alphadigits system using 32-mixture components.

relative approximation error introduced by the algorithm increased from 0.9% to 3% when the tree depth increased from 3 to 6 at a threshold of 0.4. Also, for a tree depth of 6, as the threshold increased from 0.3 to 0.8, the WER increased from 10.3% to 15.1%. Thus, the Extended BBI algorithm with a tree depth 6 and a threshold 0.3 did not introduce any additional error in the recognition system, while the algorithm with a tree depth 6 and a threshold 0.8 introduced a 46% relative approximation error.

Figure 19 shows simultaneous variations in the speedup and WER of the system as a function of the tree depth and relative threshold. In this figure, the relative threshold was

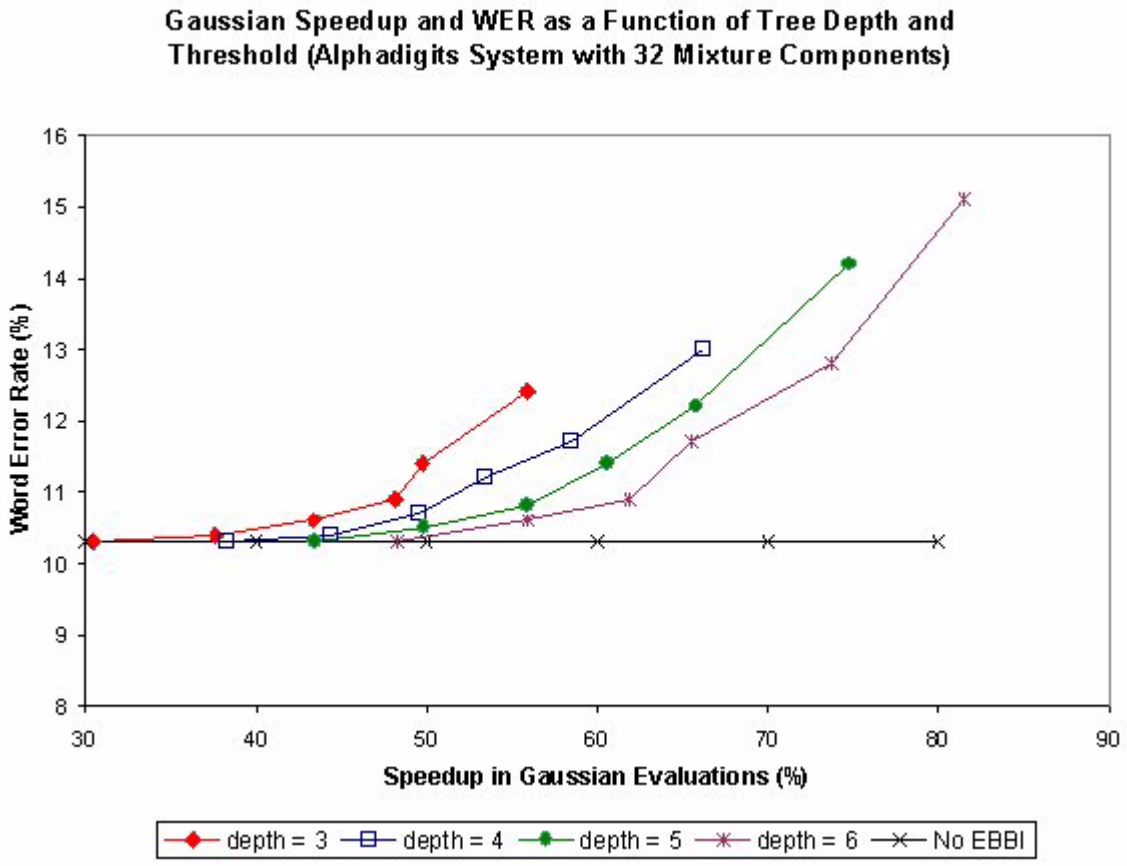

Figure 19. The simultaneous effect of the algorithm parameters on the Gaussian speedup and WER of the Alphadigits system using 32-mixture components. Different points on the plots give the Gaussian speedup and WER for a value of the tree depth and threshold.

increased from 0.3 to 0.8 along the horizontal axis. Different colors in the plot correspond to different k-d tree depths. It can be observed that for a tree depth of 6 and a threshold of 0.3, the Extended BBI algorithm produced a maximum Gaussian speedup of 48% without any increase in WER. In other words, for these values of parameters, the speed of evaluating the mixture components is almost doubled without introducing any additional error in the system.

We have seen that the algorithm produced approximately 50% Gaussian speedup without introducing any approximation error for the Alphadigits system with 32-mixture components. But, the same amount of speedup was obtained at a cost of 0.4% absolute approximation error for the Alphadigits system with 16-mixture components. Also, a tree depth of 5 with a threshold of 0.3 produced a 43% speedup without any additional error in the Alphadigits system with 32-mixture components. However, the same algorithm parameters resulted in a 41% speedup with 0.2% absolute approximation error in the Alphadigits system with 16-mixture components. Thus, we can conclude that the Extended BBI algorithm is more attractive as the number of Gaussians increases.

To gain a better understanding of the effect of the number of mixture components, we used the Alphadigits system with 64-mixture components [77]. K-d trees of depth 5, 6 and 7 were used with a relative threshold varying from 0.3 to 0.8 at steps of 0.1. The performance of the algorithm is shown in Table 9. The execution time for the evaluation of the mixture components in this system was 57.9% of the total CPU time used by the recognition system, which is 83% higher than the time consumed by the Gaussian evaluations in the system using 16-mixture components. The effect of the algorithm parameters on the Gaussian speedup is shown in Figure 20. It can be observed that the speedup in mixture component evaluations increases with an increase in the relative threshold or k-d tree depth. For a k-d tree depth of 7, the speedup in mixture component evaluations increased from 52% to 84% as the relative threshold increased from 0.3 to 0.8. Also, for a fixed threshold of 0.8, as the k-d tree depth varied from 5 to 7, the Gaussian speedup increased from 71.5% to 84%.

Tree Depth	Relative Threshold	% of Total CPU Time Taken by Gaussian Score Evaluations	Gaussian Speedup (%)	WER (%)
No EBBI System		57.88	-	10.1
5	0.3	36.00	37.80	10.2
6	0.3	32.33	44.14	10.2
7	0.3	27.90	51.80	10.3
5	0.4	32.78	43.37	10.2
6	**0.4**	**29.14**	**49.65**	**10.2**
7	0.4	24.83	57.10	10.3
5	0.5	28.66	50.48	10.6
6	0.5	25.16	56.53	10.6
7	0.5	21.77	62.39	10.7
5	0.6	25.65	55.88	11.0
6	0.6	21.62	62.65	11.2
7	0.6	18.02	68.87	11.5
5	0.7	20.98	63.75	11.6
6	0.7	16.84	70.91	12.1
7	0.7	12.96	77.61	12.4
5	0.8	16.50	71.49	12.8
6	0.8	12.57	78.28	13.6
7	0.8	9.06	84.35	14.4

Table 9. Performance of the Alphadigits system using the Extended BBI algorithm. The system used 64-mixture components.

The variation in WER is shown as a function of the algorithm parameters in Figure 21. It can be seen that for a threshold of 0.8, as the tree depth increased from 5 to 7, the Gaussian speedup increased from 12.8% to 14.4%. For a tree depth of 7, as the threshold increased from 0.3 to 0.8, WER increased from 10.2% to 14.4%. Gaussian

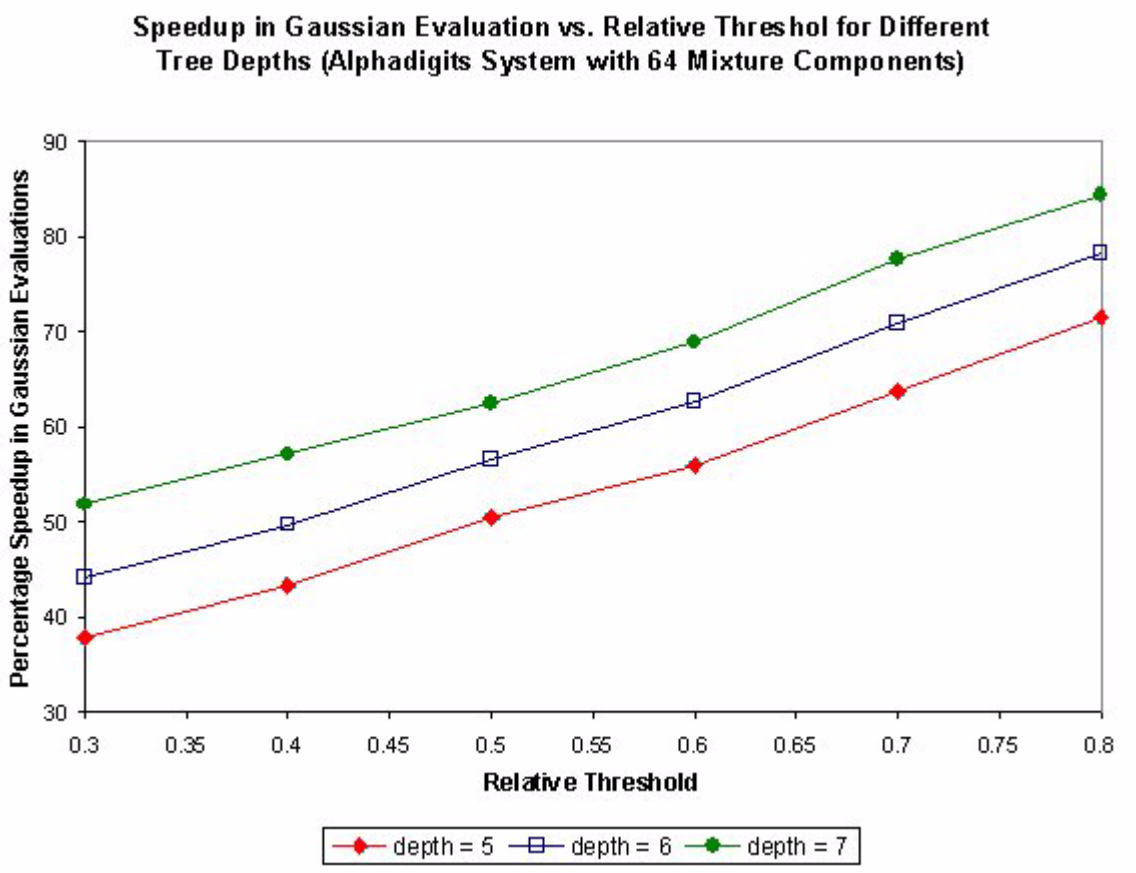

Figure 20. The effect of the algorithm parameters on the speedup in Gaussian evaluations for an Alphadigits system using 64-mixture components.

speedup and WER are simultaneously plotted against algorithm parameters in Figure 22. We can observe that for a tree depth of 6 and a threshold of 0.4, the speed of Gaussian evaluations is almost doubled with only 0.1% absolute or 0.9% relative increase in WER. Since this is the maximum amount of speedup obtained with the smallest approximation error, these parameters can be considered to be the optimal algorithm parameters.

We can conclude that for the systems with higher mixture components, the same amount of speedup can be obtained with a significantly lower increase in WER. Table 10 gives the relative approximation error introduced by the algorithm to obtain a 60% and a

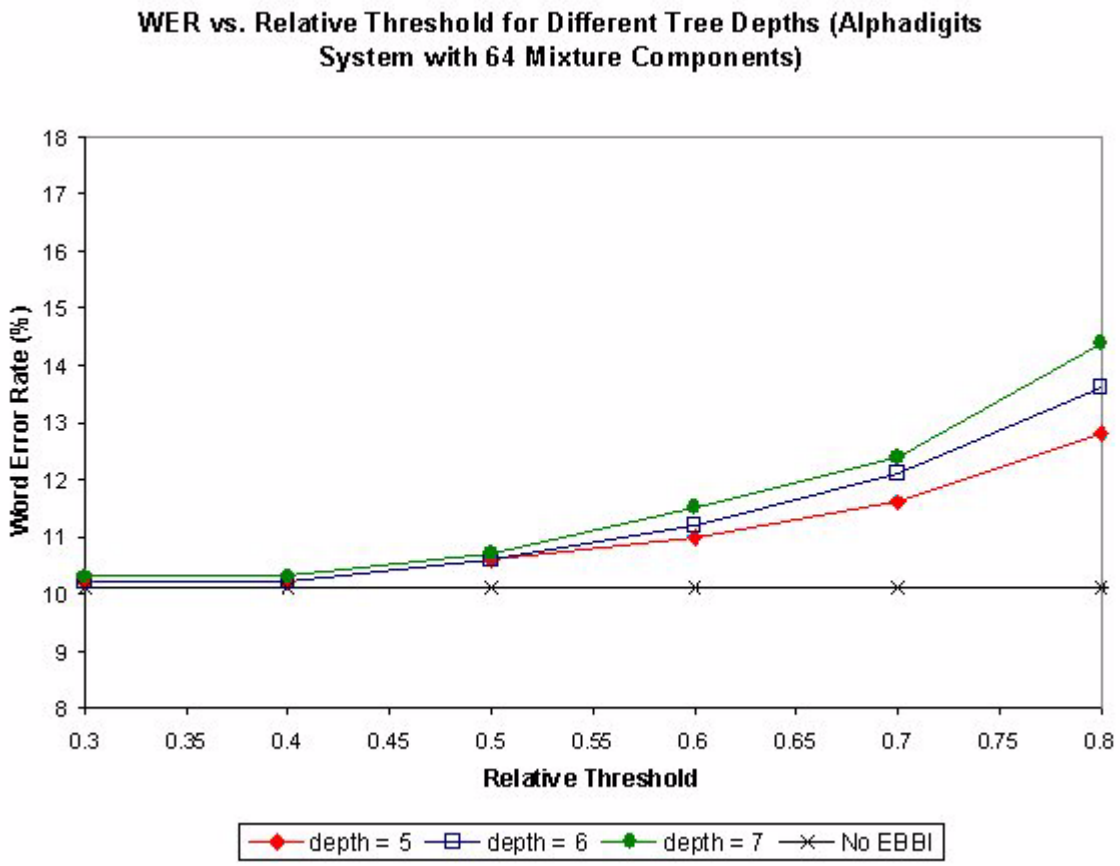

Figure 21. Variations in WER as a function of the algorithm parameters for an Alphadigits system using 64-mixture components.

80% Gaussian speedup for the Alphadigits system with 16, 32 and 64 mixture components. In the Alphadigits system with 16-mixture components, the Extended BBI algorithm introduced a 16.5% relative approximation error to yield a speedup of 60%. On the other hand, the algorithm produced the same amount of speedup with only about 3% approximation error when the Alphadigits system with 64-mixture components was used.

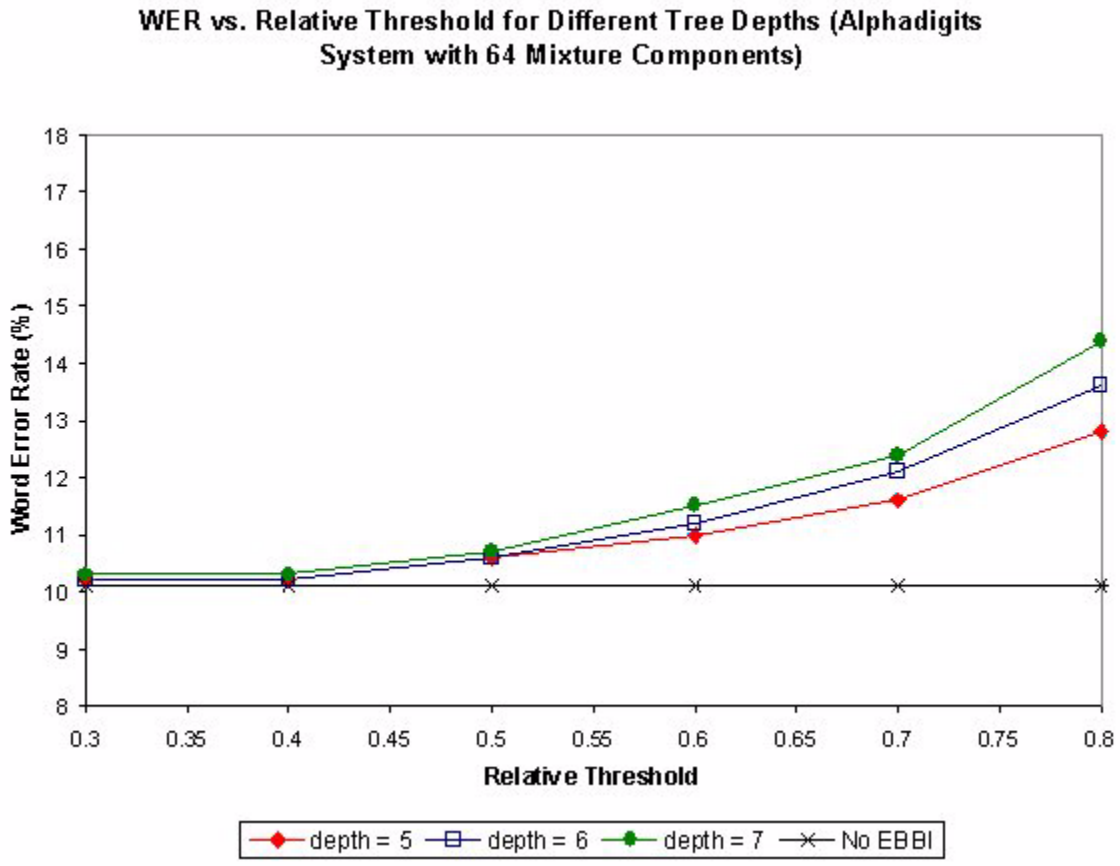

Figure 22. The effect of algorithm parameters on the Gaussian speedup and WER for an Alphadigits system using 64-mixture components.

Speedup in Gaussian Evaluations (%)	**Relative Approximation Error Introduced by Extended BBI Algorithm (%)**		
	16-mixture	**32-mixture**	**64-mixture**
60	16.50	4.85	2.97
80	51.45	46.60	32.67

Table 10. The approximation error introduced by the Extended BBI algorithm to obtain a certain Gaussian speedup for the Alphadigits system with varying mixture components.

SWITCHBOARD

SWITCHBOARD is a more complex task than the TIDIGITS and the OGI Alphadigits tasks [78]. A set of experiments, with an error threshold varying from 0.3 to 0.8 and the tree depth varying from 3 to 5, were run. The performance of the algorithm is given in Table 11. It can be observed that the time consumed by the evaluation of mixture Gaussians decreases as the tree depth or relative threshold increases. Also, the approximation error introduced by fast and approximative computations increases with an increase in tree depth or relative threshold.

The effect of the algorithm parameters on the Gaussian speedup is shown in Figure 23. Different colors in the plot correspond to different k-d tree depths. Figure 24 shows the variations in WER of the system as a function of the algorithm parameters. As the relative threshold increases, the speedup in Gaussian evaluations increases at a cost of a higher WER. This behavior is consistent with the behavior of the algorithm on the other tasks. For the SWB task, the Extended BBI algorithm doubled the speed of score computation with only a 3.4% relative increase in the WER. Figure 25 gives the simultaneous effect of the algorithm parameters on the Gaussian speedup and WER of the system. We can observe that a speedup of 42% in mixture component evaluations was obtained with only a 1.2% relative increase in the WER.

4.7. Gaussian Clipping and Extended BBI Algorithm

Another technique that has been used in this thesis to speed up the evaluation of the mixture Gaussians is known as Gaussian clipping. It is used to generate lower-order

Tree Depth	Relative Threshold	% of Total CPU Time Taken by Gaussian Score Evaluations	Gaussian Speedup (%)	WER (%)
No EBBI System		9.92	-	41.1
3	0.3	8.04	18.95	41.3
4	0.3	7.88	20.56	41.3
5	0.3	7.70	22.38	41.3
3	0.4	7.28	26.61	41.4
4	0.4	7.05	28.93	41.4
5	0.4	6.78	31.65	41.4
3	0.5	6.61	33.37	41.5
4	0.5	6.15	38.00	41.5
5	0.5	5.75	42.04	41.6
3	0.6	5.93	40.22	42.0
4	0.6	5.29	46.67	42.2
5	**0.6**	**4.89**	**50.70**	**42.5**
3	0.7	5.06	48.99	42.6
4	0.7	4.34	56.25	43.1
5	0.7	3.77	61.99	43.8
3	0.8	4.32	56.45	43.5
4	0.8	3.50	64.72	44.4
5	0.8	2.89	70.87	45.6

Table 11. The performance of the Extended BBI algorithm as a function of the tree depth and relative threshold for an SWB task.

mixture models from higher-order mixture models by removing Gaussians with low mixture weights. Mixture weights were re-normalized in this technique. For a system with 64-mixture components, this technique gave a speedup of 50% by using 32 Gaussians components with higher mixture weights.

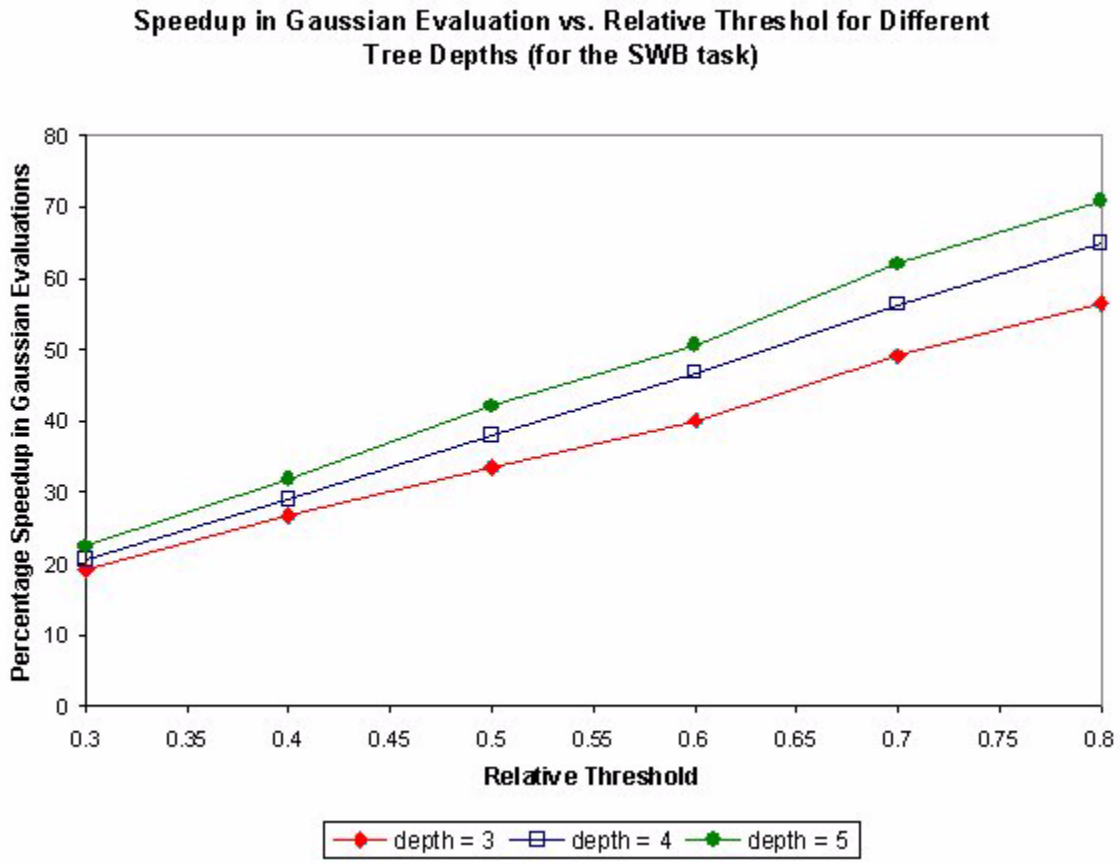

Figure 23. The speedup in Gaussian evaluations as a function of the relative threshold and k-d tree depth for an SWB system.

The experiments were run to compare the performance of the Gaussian clipping and the Extended BBI algorithm. Table 12 shows the performance of the following systems on the OGI Alphadigits task:

- No FGC — A recognition system with a lower number of mixture components (8 and 12 in this case) and no fast Gaussian computation techniques;
- Extended BBI (16-mixture) — A recognition system using the Extended BBI algorithm to extract 8 and 12 mixture components from 16 mixture components;
- Gaussian Clipping (16-mixture) — A recognition system using the Gaussian clipping technique to extract 8 and 12 mixture components from 16 mixture components.

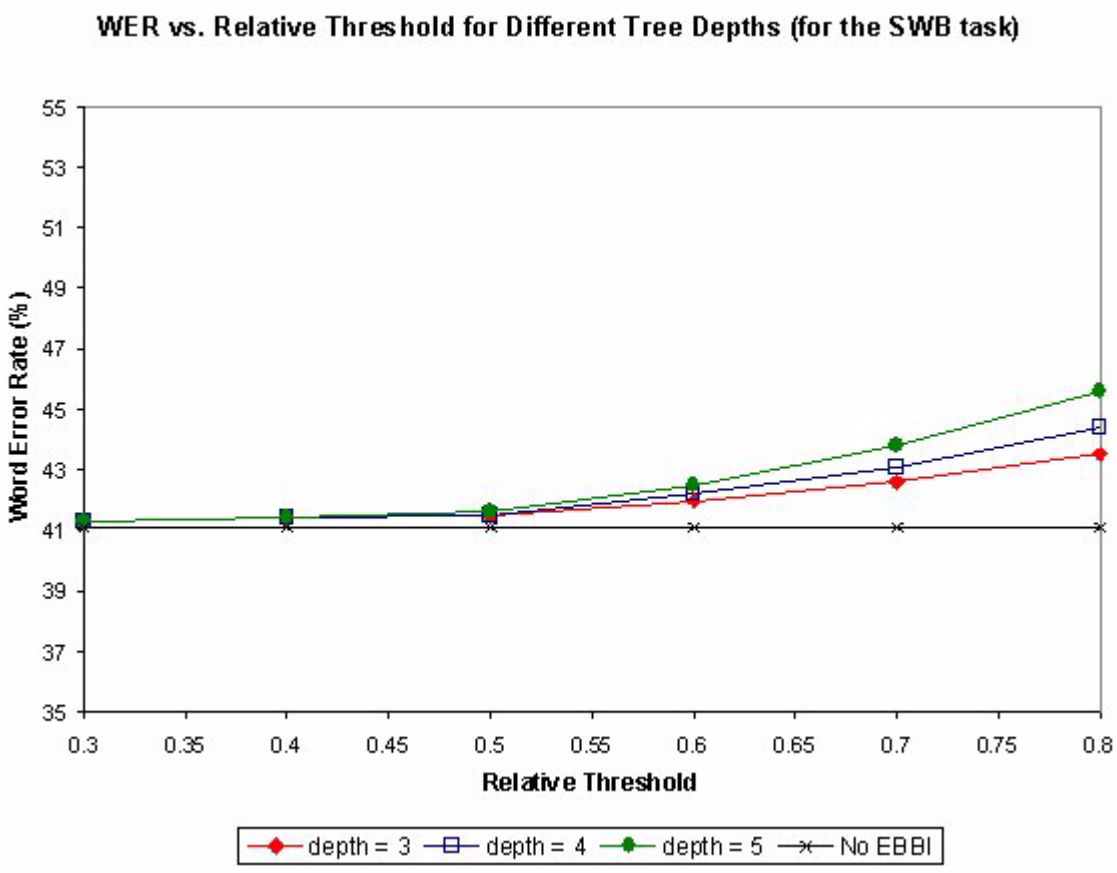

Figure 24. The effect of the algorithm parameters on the WER for an SWB system.

From the results shown in Table 12, it can be observed that the "No FGC" system using 8-mixture components produced a WER of 11.6%. When the Gaussian clipping approach is used to extract 8 Gaussian components, the system gave a WER of 12.2%. Thus, the Gaussian clipping technique results in a degradation of system performance. When the Extended BBI algorithm was used to extract 8 Gaussians, a WER of 10.7% was obtained. This suggests that the performance of the system using the Extended BBI algorithm is 8% better than the "No FGC" system, and 12.3% better than the Gaussian clipping system.

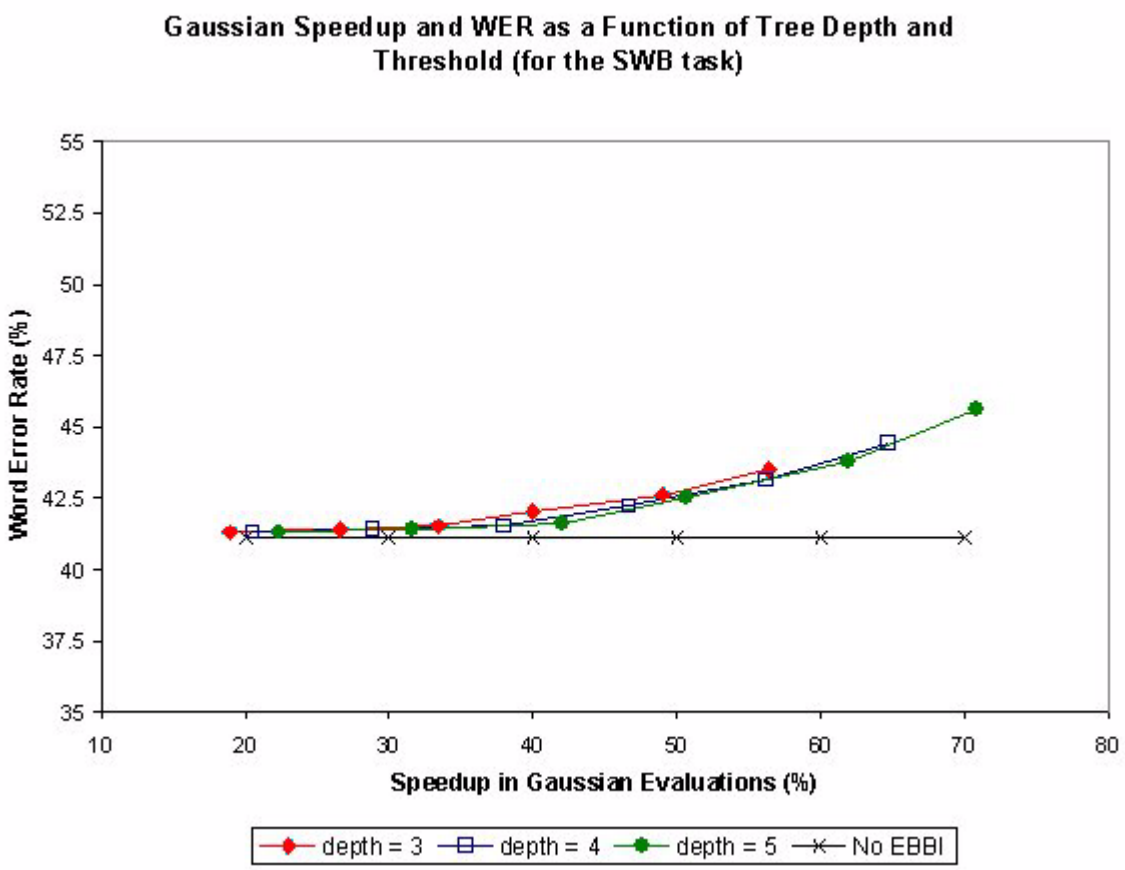

Figure 25. The effect of the algorithm parameters on the Gaussian speedup and WER for an SWB task.

Number of Mixtures	Technique Used	WER (%)
8-mixture	No FGC	11.6
	Extended BBI (16-mix)	10.7
	Gaussian Clipping (16-mix)	12.2
12-mixture	No FGC	10.7
	Extended BBI (16-mix)	10.3
	Gaussian Clipping (16-mix)	11.5

Table 12. A comparison of performance for several variants of the fast Gaussian computational approach.

The experimental results gives us enough evidence to conclude that the Extended BBI algorithm is superior to the Gaussian clipping technique. We can also conclude that the performance of the system using the Extended BBI algorithm is better than the system using fewer mixture components and no Gaussian speedup techniques.

CHAPTER 5

CONCLUSIONS AND FUTURE DIRECTIONS

This thesis investigated major drawbacks of the BBI algorithm and proposed an improvement of the algorithm known as the Extended BBI algorithm. The algorithm has been applied to current state-of-the-art LVSCR system and it was shown via experimentation that the Extended BBI algorithm is superior to the BBI algorithm and Gaussian clipping technique.

5.1. Thesis Contributions

Extended BBI Algorithm and BBI Algorithm

We found that the optimization criterion used by the BBI algorithm failed to provide an optimal hyperplane in the case of multiple minima. The Extended BBI algorithm proposed a modified optimization criterion which provided optimal splits in case of multiple minima. The optimization criterion used by the Extended BBI algorithm produced a 10% lower WER than the optimization criterion used by the BBI algorithm. Re-normalization of mixture weights used in the Extended BBI algorithm reduced the WER by 3%. The proposed algorithm produced a 12% lower WER than the BBI algorithm, without reducing Gaussian speedup.

Gaussian Clipping and Extended BBI Algorithm

The Extended BBI algorithm has also been compared to the Gaussian clipping technique for speeding up mixture component evaluations. Experimental results showed that the Extended BBI algorithm produced a 12% lower WER than the Gaussian clipping technique. Both algorithms were evaluated for the case of a 50% speedup in mixture component evaluations on the Alphadigits task using 16-mixture components.

5.2. Summary of Experiments

We evaluated the performance of the Extended BBI algorithm on small as well as large vocabulary tasks. For small vocabulary, TIDIGITS and OGI Alphadigits tasks were used. For the large vocabulary task, we used the SWB conversational speech corpus. The performance of the algorithm was also evaluated for a range of mixture component orders.

For TIDIGITS, the Extended BBI algorithm produced a speedup of 45% without any increase in WER. On the same task, a speedup of approximately 58% was obtained with only a 0.1% absolute increase in WER. For the Alphadigits system using 32-mixture components, the Extended BBI algorithm produced a speedup of 48%. Thus, the algorithm allows twice the number of mixture components to be used. This results in a reduced WER since error rates generally decrease as the number of mixture components are increased. For example, the Alphadigits system using 32-mixture components gives a 10.3% WER and the Alphadigits system using 64-mixture components gives a 10.1% WER. For the SWITCHBOARD task, the Extended BBI algorithm achieved a speedup of 42% with a 1.2% relative increase in WER of the system.

5.3. Future Work

We have verified that the modified optimization criterion used by the Extended BBI algorithm yields a significant improvement over the BBI algorithm. However, we still need to address some issues related to the optimization criterion.

The optimization criterion used by the Extended BBI algorithm focuses on building a balanced binary tree by obtaining a separating hyperplane perpendicular to one of the coordinate axis. In real applications this optimization criterion results in more than one such hyperplanes. The Extended BBI algorithm computes the mean of the positions of two such extreme hyperplanes and uses the hyperplane at the mean as division hyperplane. It is important to study the performance of the Extended BBI algorithm under several balanced hyperplanes. This is because the k-d trees obtained by using different hyperplanes may result in a different set of most significant Gaussians.

The Extended BBI algorithm did not consider mixture weights in computing the Gaussian boxes. This is because the mixture weights do not significantly change the Gaussians with large likelihoods. The mixture weights can be included in the computation of the Gaussian boxes, and this should provide an improvement in performance without an additional computational cost during recognition.

REFERENCES

[1] F. Jelinek, R. Mercer and S. Roukos, “Principles of Lexical Modeling for Speech Recognition,” *Advances in Speech Signal Processing*, pp. 651-699, Marcel Dakker Inc., New York, New York, USA, 1992.

[2] J. Picone, “LVCSR Systems: Challenging the Limits of Computing,” *http://www.isip.msstate.edu/conferences/srstw01/program/session_10/lvcsr/index.html*, Mississippi State, Mississippi, USA, May 2001.

[3] J. Odell, P. Woodland and T. Hain, “The CUHTK-Entropic 10xRT Broadcast News Transcription System,” *Proceedings of DARPA Broadcast News Workshop,* Herndon, Virginia, USA, Feb. 1999.

[4] S. Matsoukas, L. Nguyen, J. Davenport, J. Billa, F. Richardson, M. Siu, D. Liu, R. Schwartz and J. Makhoul, “The 1998 BBN BYBLOS Primary System applied to English and Spanish Broadcast News Transcription,” *Proceedings of DARPA Broadcast News Workshop,* Herndon, Virginia, USA, Feb. 1999.

[5] G.F. Chollet and C. Gagnoulet, “On the Evaluation of Speech Recognizers and Databases Using a Reference System,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 2026-2029, Paris, France, 1982.

[6] M. Gales, K. Knill and S. Young, “State-Based Gaussian Selection in Large Vocabulary Continuous Speech Recognition Using HMM's,” *IEEE Transactions on Speech and Audio Processing*, Vol. 7, No. 2, pp. 152-161, March 1999.

[7] P. Woodland, T. Hain, G. Moore, T.R. Niesler, D. Povey, A. Tuerk and E. Whittaker, “The 1998 HTK Broadcast News Transcription System: Development and Results,” *Proceedings of DARPA Broadcast News Workshop,* Herndon, Virginia, USA, Feb. 1999.

[8] P. Woodland, T. Hain, G. Moore, T. Niesler, D. Povey, A. Tuerk and E. Whittaker, “The 1998 BBN Byblos 10X Real Time System,” *Proceedings of DARPA Broadcast News Workshop,* Herndon, Virginia, USA, Feb. 1999.

[9] P. Woodland, T. Hain, G. Evermann and D. Povey, “CU-HTK March 2001 Hub5 System,” *2001 Large Vocabulary Conversational Speech Recognition Workshop*, Baltimore, Maryland, May 2001.

[10] F. Jelinek, *Statistical Methods for Speech Recognition*, MIT Press, Cambridge, Massachusetts, USA, 1997.

[11] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.

[12] A. Robinson, “An Application of Recurrent Nets to Phone Probability Estimation,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 298-305, March 1994.

[13] J. Picone, “Continuous Speech Recognition Using Hidden Markov Models,” *IEEE Acoustics, Speech, and Signal Processing Magazine*, vol. 7, no. 3, pp. 26-41, July 1990.

[14] B. Necioglu, M. Ostendorf and J. Rohlicek, “A Bayesian Approach to Speaker Adaptation for the Stochastic Segment Model,” *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. I, pp. 437-440, San Francisco, California, March 1992.

[15] X. Huang, H. W. Hon, M. Hwang and K. Lee, “A Comparative Study of Discrete, Semi-Continuous and Continuous Hidden Markov Models,” *Computer Speech and Language*, vol. 7, no. 4, pp. 359-368, October 1993.

[16] J. Deller, J. Proakis and J. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillan Publishing, New York, USA, 1993.

[17] H. Ney, U. Essen and R. Kneser, “On Structuring Probabilistic Dependencies in Stochastic Language Modeling,” *Computer Speech and Language*, vol. 8, no. 1, pp. 1-38, January 1994.

[18] D. Paul, “An Essential A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, San Francisco, California, USA, March 1992.

[19] N. Deshmukh, A. Ganapathiraju and J. Picone, “Hierarchical Search for Large Vocabulary Conversational Speech Recognition,” *IEEE Signal Processing Magazine*, vol. 16, no. 5, pp. 84-107, September 1999.

[20] J. Odell, V. Valtchev, P. Woodland and S. Young, “A One Pass Decoder Design for Large Vocabulary Recognition,” *Proceedings of ARPA Human Language Technology Workshop*, pp. 405-410, Princeton, New Jersey, USA, March 1994.

[21] H. Murveit, J. Butzberger, V. Digalakis and M. Weintraub, “Progressive-Search Algorithms for Large Vocabulary Speech Recognition,” *Proceedings of the DARPA Human Language Technology Workshop*, Cambridge, Massachusetts, USA, March 1993.

[22] N. Deshmukh, A. Ganapathiraju, J. Hamaker and J. Picone, “An Efficient Public Domain LVCSR Decoder,” *Proceedings of the Hub-5 Conversational Speech Recognition (LVCSR) Workshop*, Linthicum Heights, Maryland, USA, September 1998.

[23] J. Picone, “Signal Modeling Techniques in Speech Recognition,” *Proceedings of the IEEE*, vol. 81, no. 9, pp. 1215-1247, September 1993.

[24] H. Hermansky, “Perceptual Linear Predictive (PLP) Analysis of Speech,” *Journal of the Acoustical Society of America*, vol. 87, no. 4, pp. 1738-1752, April 1990.

[25] S. Furui, “Cepstral Analysis Technique for Automatic Speaker Verification,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 2, pp. 254-272, April 1981.

[26] R. Bahl et al, “Large Vocabulary Natural Language Continuous Speech Recognition,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 465-467, Glasgow, Scotland, May 1989.

[27] Intel Developer Services, “An Overview of Speech Technology and Applications,” *http://developer.intel.com/software/idap/resources/technical_collateral/pentiumii/speech.htm*, September 1998.

[28] M. Padmanabhan, E. Jan, L. Bahl and M. Picheny, “Decision-tree based feature-space quantization for fast gaussian computation,” *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop*, pp. 325-330, Santa Barbara, California, USA, December 1997.

[29] J. Godfrey, E. Holliman, and J. McDaniel, “SWITCHBOARD: Telephone Speech Corpus for Research and Development,” *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, vol. 1, pp. 517-520, San Francisco, California, USA, March 1992.

[30] L. Bahl, P. deSouza, P. Gopalakrishnan, D. Nahamoo and M. Picheny, “Robust Methods for using Context-Dependent Features and Models in a Continuous Speech Recognizer,” *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 533-536, Adelaide, Australia, April 1994.

[31] J. Odell, *The Use of Context in Large Vocabulary Speech Recognition*, Ph. D. Thesis, Cambridge University, UK, 1997.

[32] R. Leonard, “A Database for Speaker-Independent Digit Recognition,” *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, 1984.

[33] A. Poritz, “Hidden Markov Models: A Guided Tour,” *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 7-13, New York City, New York, USA, April 1998.

[34] L. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proceedins of the IEEE*, vol. 37, no. 2, pp. 257-286, February 1989.

[35] L. Baum and T. Petrie, “Statistical Inference for Probabilistic Functions of Finite State Markov Chains,” *Annals of mathematical statistics*, vol. 37, pp. 1559-1563, 1966.

[36] P. Woodland and D. Cole, “Optimizing Hidden Markov Models using Discriminative Output Distributions,” *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, April 1991.

[37] L. Bahl, P. Brown, P. deSouza and R. Mercer, “Estimating Hidden Markov Model Parameters so as to Maximize Speech Recognition Accuracy,” *IEEE Transactions on Speech and Audio Processing*, vol. 1, no. 1, part II, pp. 77-83, January 1993.

[38] J. Picone, “Fundamentals of Speech Recognition,” *http://www.isip.msstate.edu/publications/courses/isip_0000/*, April 1998.

[39] M. Woszczyna, *Fast Speaker Independent Large Vocabulary Continuous Speech Recognition*, Ph.D. dissertation, Karlsruhe, Germany, February 1998.

[40] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, San Diego, CA, USA, 1990.

[41] M. Ostendorf, “From HMMs to Segment Models: A Unified View of Stochastic Modeling for Speech Recognition,” *IEEE Transactions on Speech and Audio Processing*, vol. 4, no. 5, part II, pp. 360-378, September 1996.

[42] S. Levinson, L. Rabiner and M. Sondhi, “An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition,” *Bell System Technical Journal*, vol. 62, no. 4, pp. 1035-1074, 1983.

[43] E. Bocchieri, “Vector quantization for efficient computation of continuous density likelihoods,” *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 692-695, Minneapolis, USA, 1993.

[44] H. Murveit, P. Monaco, V. Digalakis and J. Butzberger, “Techniques to achieve an accurate real-time large-vocabulary speech recognition system,” *Proceedings of the ARPA Workshop on Human Language Technology,* pp. 368-373, Austin, Texas, USA, March 1995.

[45] S. Arya, “Algorithms for Fast Vector Quantization,” *Proceedings of the IEEE Data Compression Conference, pp.* 381-390, Snowbird, Utah, USA, March 1993.

[46] W. Equitz, “A New Vector Quantization Clustering Algorithm,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 37, no. 10, pp. 1568-1575, New Paltz, New York, USA, October 1989.

[47] W. Equitz, “Fast Algorithms for Vector Quantization Picture Coding,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 725-728, 1987.

[48] J. Friedman, J. Bentley and R. Finkel, “An Algorithm for Finding Best Matches in Logarithmic Expected Time,” *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209-226, September 1977.

[49] J. Makhoul, S. Roucos and H. Gish, “Vector Quantization in Speech Coding,” *Proceedings of the IEEE*, vol. 73, no. 11, pp. 1551-1588, New York, USA, November 1985.

[50] C. Bei and R. Gray, “An Improvement of the Minimum Distortion Encoding Algorithm for Vector Quantization,” *IEEE Transactions on Communications*, vol. 33, pp. 1132-1133, October 1985.

[51] L. Fissore, P. Laface, P. Massafra, and F. Ravera, “Analysis and Improvement of the Partial Distance Search Algorithm,” *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 315-318, 1993.

[52] D. Cheng and A. Gersho, “A fast codebook search algorithm for nearest-neighbor pattern matching,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 265-268, Tokyo, April 1986.

[53] J. Fritsch, I. Rogina, T. Sloboda, A. Waibel, “Speeding Up The Score Computation Of HMM Speech Recognizers With The Bucket Voronoi Intersection Algorithm,” *Proceeding of the EUROSPEECH,* vol 2, pp. 1091-1094, Madrid, Spain, 1995.

[54] J. Fritsch, and I. Rogina, “The Bucket Box Intersection (BBI) Algorithm for fast approximative evaluation of Diagonal Mixture Gaussians,” *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 837-840, 1996.

[55] J. Bentley, “Multidimensional Binary Search Trees Used for Associative Searching,” *Communications of the ACM,* vol. 18, no. 9, pp. 509-517, September 1975.

[56] J. Baker, *Stochastic Modeling for Automatic Speech Understanding*, Academic Press, New York, USA, 1975.

[57] A. Lowry, S. Hossain and W. Millar, “Binary Search Trees for Vector Quantization,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 2205-2208, Dallas, Texas, USA, May 1987.

[58] J. Bentley, “Multidimensional Binary Search Trees in Database Applications,” *IEEE Transactions on Software Engineering,* vol. 5, no. 4, pp. 333-340, New York, USA, July 1979.

[59] V. Ramasubramanian and K. Paliwal, “A generalized Optimization of the K-d Tree for Fast Nearest Neighbor Search,” *Proceedings of 4th IEEE Region 10 International Conference on TENCON*, pp. 565-568, November 1989.

[60] V. Ramasubramanian and K. Paliwal, “An Optimized K-d Tree Algorithm for Fast Vector Quantization of Speech,” *Signal Processing IV: Theories and Applications,* pp. 875-878, North Holland, 1988.

[61] K. Paliwal and V. Ramasubramanian, “Effect of Ordering the Codebook on the Efficiency of the Partial Distance Search for Vector QUantization,” *IEEE Transactions on Communications,* vol. 37, pp. 538-540, May 1989.

[62] H. Zhao et al, “Multidimensional Searching Trees with Minimum Attribute,” *JSSST Computer Software*, vol. 19, no.1, pp.22-28, Ooita-ken, Japan, 2002.

[63] S. Nene and S. Nayar, “A Simple Algorithm for Nearest Neighbor Search in High Dimensions,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 19, no. 9, pp. 989-1003, September 1997.

[64] L. Devroye et al, “Squarish k-d trees,” *Society for Industrial and Applied Mathematics* (*SIAM) Journal on Computing,* vol. 30, no. 5, pp. 1678-1700, 2000.

[65] I. Al-Furaih, “Parallel Construction of Multidimensional Binary Search Trees,” *IEEE Transactions on Parallel and Distributed Systems,* vol. 11, no. 2, pp. 136-148, February 2000.

[66] P. Chanzy, L. Devroye, C. Zamora-Cura, “Analysis of range search for random k-d trees,” *Acta Informatica*, vol. 37, no. 3, pp. 355-383, 2001.

[67] R. Cole et. al., “Alphadigit Corpus,” *http://www.cse.ogi.edu/CSLU/corpora/alphadigit,* Center for Spoken Language Understanding, Oregon Graduate Institute, 1997.

[68] Center for Spoken Language Understanding (CSLU), “Alphadigit v1.1,” *http://cslu.cse.ogi.edu/corpora/alphadigit/,* September 1996.

[69] D. Deterding, et. al., “Vowel Recognition”, available at *http://www.ics.uci.edu/pub/machine-learning-databases/undocumented/connectionist-bench/vowel/,* August 2000.

[70] J. Godfrey, E. Holliman, and J. McDaniel, “SWITCHBOARD: Telephone Speech Corpus for Research and Development,” *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, vol. 1, pp. 517-520, San Francisco, California, USA, March 1992.

[71] J. Godfrey and E. Holliman, “SWITCHBOARD-1 Release 2,” available at *http://www.ldc.upenn.edu/Catalog/LDC97S62.html,* September 1993.

[72] E. Shriberg, “Disfluencies in SWITCHBOARD,” *Proceedings of the International Conference on Spoken Language Processing,* Vol. Addendum, pp. 11-14, Philadelphia, PA, October 1996.

[73] J. Fenlason and R. Stallman, “The GNU Profiler,” available at *http://www.gnu.org/manual/gprof-2.9.1/html_mono/gprof.html*, November 1998.

[74] N. Deshmukh, et. al., “A Public Domain Speech-to-Text System,” *Proceedings of. Eurospeech*, vol. 5, Budapest, Hungary, September 1999.

[75] J. Hamaker, A. Ganapathiraju, J. Picone and J. Godfrey, “Advances in Alphadigit Recognition Using Syllables,” *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 421-424, Seattle, Washington, USA, May 1998.

[76] J. Hamaker, et al, “A Proposal for a Standard Partitioning of the OGI AlphaDigit Corpus,” available at *http://isip.msstate.edu/projects/lvcsr/recognition_task/alphadigits/data ogi_alphadigits/trans_eval.text*.

[77] P. Loizou and A. Spanias, “High-Performance Alphabet Recognition,” *IEEE Transactions on Speech and Audio Processing*, pp. 430-445, November 1996.

[78] E. Shriberg, “Phonetic Consequences of Speech Disfluency,” *Symposium on The Phonetics of Spontaneous Speech — Proceedings of the International Congress of Phonetic Sciences*, vol. 1, pp. 619-622, San Francisco, CA, 1999.

[79] P. Woodland, T. Hain, G. Evermann and D. Povey, “CU- HTK March 2001 Hub5 System,” *2001 Large Vocabulary Conversational Speech Recognition Workshop*, Baltimore, Maryland, May 2001.

[80] L. Gillick and S. Cox, “Some Statistical Issues in the Comparison of Speech Recognition Algorithms,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 532-535, Glasgow, May 1999.

[81] Sun Microsystems, “Profiling Programs with prof, gprof, and tcov,” available at *http://docs.sun.com/source/816-2458/OtherTools.html#pgfId-997973*, February 2002.

[82] Rational Software Corporation, “Using Rational Quantify,” available at *http://amwdb.u-strasbg.fr/docs/purify/html/installing_and_gettingstarted/4-quantify.html*, January 2001.

[83] Rice Computer Science, “Performance Profiling,” available at *http://www.owlnet.rice.edu/~comp320/2003/notes/tut10-profiling/*, August 2002.

APPENDIX A

This appendix describes various profiling tools and provides a detailed analysis of the gprof tool that is used in this thesis. There are several industry-standard tools used for performance profiling. Two simple UNIX tools are *time* and *top*. But neither of these can show the resource usage of the individual modules or functions. Other industry-leading tools are *tcov, Quantify, prof* and *gprof* [81]. The tcov tool counts the number of time each function is executed but it does not give the execution time for each of the functions. The Quantify tool uses an Object Code Insertion (OCI) technology to count the instructions a program executes and to compute how many cycles they require to execute [82]. The prof and gprof tools provide the run-time (in CPU seconds) of a program and divide this time among all functions. The gprof utility provides more features than the prof utility. Since we need a tool to find the time used by the individual functions of a program, we can use the Quantify and gprof tools for profiling. In this thesis, we have used Intel-based UNIX systems and Quantify is only available for Sun Sparc architectures. Therefore, we have used the gprof utility.

The output of gprof is a file containing two tables, the *flat profile* and the *call graph*. The flat profile gives the total execution times and call counts for each of the functions in the program. The call graph shows how much time was spent in each function

and its children. In order to understand the accuracy of gprof, we ran the recognizer on a small set of utterances and used the gprof utility to find the execution time of the Gaussian computations. We ran the recognizer several times using different load conditions on the same machine. Table 13 gives the results of 10 test runs. Using these results, the mean execution time is found to be 16.68 seconds and the standard deviation is found to be 0.2, which gives a 2.5 digits of accuracy.

We can observe that there is a small difference in the execution time of the Gaussian evaluations between different test runs. This is because the run-time figures provided by gprof are based on a sampling process [83]. The rule of thumb is that a run-time figure is accurate if it is considerably larger than the sampling period. By default, the gprof uses a sampling period of 0.01 seconds. Since the run-time of the recognizer is considerably larger than the sampling period, the run-time figures obtained in this thesis are accurate. In case of smaller run-time figures, we can get a higher accuracy by combining the data from several test runs, using the *-s* option of gprof [73].

Run Number	**Execution Time of Gaussian Computations (Seconds)**	**Run Number**	**Execution Time of Gaussian Computations (Seconds)**
1	16.77	6	16.79
2	16.23	7	16.91
3	16.79	8	16.85
4	16.56	9	16.59
5	16.48	10	16.80

Table 13. Execution time of the Gaussian computations for several test runs.