Improvements to Bucket Box Intersection Algorithm for Fast GMM Computation in Embedded Speech Recognition Systems

Min Tang, Aravind Ganapathiraju

Conversay
Redmond, WA 98052, USA
{mtang, aganapat}@conversay.com

10.21437/Interspeech.2006-223

Abstract

Real-time performance is a very important goal for embedded speech recognition systems, where the evaluation of likelihoods for Gaussian mixture models (GMM) usually dominates the computation of a continuous density hidden Markov model (CDHMM) based system. The Bucket Box Intersection (BBI) algorithm is an optimization technique that uses a K-Dimensional binary tree to speed up the score computation of GMM without significantly hurting the recognition accuracy. In this paper, we propose three improvements to the traditional BBI algorithm. First, we define the optimal dividing hyper-plane as the plane that generates minimum expected number of mixture evaluations instead of the median hyper-plane. The size of BBI tree is reduced largely because of that. Second, we refine the location of dividing plane as the one that has the same Mahalanobis distance to the closest dividing mixture pair, instead of the boundary of Gaussian box. By doing this, we are able to improve recognition accuracy. Finally, we introduce the dividing planes which run across two dimensions to boost the range of dividing plane candidates and thus bring more speed-ups. We evaluated these techniques using Conversay's speech engine CASSI in 2 different domains. The experimental results of new BBI algorithm show significant performance improvement over traditional BBI algorithm. Compared to the baseline system with no BBI algorithm implementation, we were able to speed up Gaussian computations by 50% with a less than 5% relative increase in word error rate.

Index Terms: embedded speech recognition, fast Gaussian computation, Bucket Box Intersection algorithm.

1. Introduction

Embedded speech recognition systems have two important goals: real-time performance and low resource usage. For the first goal, it is well known that the evaluation of likelihoods for Gaussian mixture models (GMM) usually dominates the computations in a continuous density hidden Markov model (CDHMM) based system. Typically, these computations can take anywhere from 30%-70% of the overall recognition time. Therefore, a lot of techniques have been proposed to speed up the GMM computation procedure [1]. For example, [2] uses Best Mixture Prediction and Feature Component Reordering (FCR) in a partial distance elimination (PDE) framework to reduce computational time in nearest-neighbor based search without losing any accuracy. In [3], context-independent (CI) GMM-based GMM selection was proposed to reduce GMM computation by only applying context-dependent (CD) GMM computation to the GMMs whose corresponding CI scores are higher than a beam. [4] uses Vector-Quantization (VQ) based Gaussian Selection (GS) to approximate the full score by only computing parts of full Gaussians that were considered to dominate the likelihood computation. And in [5], the approach of Bucket Box Intersection (BBI) algorithm was proposed to approximate the Gaussian score by only computing some of the most significant Gaussians which are dynamically determined based on a pre-computed k-dimensional feature space partitioning binary tree. Since BBI algorithm only optimizes the data organization of the model, it has the potential to combine with other fast GMM evaluation techniques. Furthermore, because BBI is a general technique to speed up nearest-neighbor search, it can be used in any application where vector quantization coding is required, for example, Vector-Quantization based Gaussian Selection.

In this paper, we propose three improvements to the traditional BBI algorithm [5, 6]. First, we define the optimal hyper-plane as the plane that generates minimum expected number of mixture evaluations instead of the median hyper-plane. This optimal hyper-plane determination criteria reduces the BBI tree by a large amount, resulting in low resource usage. Second, we optimize the location of the dividing plane as the one that has the same Mahalanobis distance to the closest dividing mixture pair, instead of one the two boundary of Gaussian box. By doing this, we are able to improve recognition accuracy without additional memory and computation cost. Finally, we introduce cross-dimensional dividing plane which gives bigger range to dividing plane candidates and thus gives us more speed-up. Evaluation is done using Conversay's embedded speech engine – CASSI in 2 domains – Aurora noisy environment digits recognition and Samsung cellphone command-and-control recognition. The experimental results show significant performance improvement over traditional BBI algorithm. Compared to traditional BBI algorithm, we were able to reduce approximation errors by 50% with the trees 1/4 the size of those in the traditional BBI algorithm.

2. Bucket Box Intersection Algorithm

BBI algorithm reduces the feature search space by using a pre-computed K-dimensional binary partitioning tree in nearest-neighbor based search framework.

2.1. Nearest-Neighbor Approximation Framework

The likelihood of one Gaussian mixture model λ for a given feature vector $\vec{x_t}$ can be computed as:

$$L(\vec{x_t}|\lambda) = \sum_{m=1}^{M} p_m \frac{exp\{-\frac{1}{2}(\vec{x_t} - \vec{\mu_m})' \Sigma_m^{-1} (\vec{x_t} - \vec{\mu_m})\}}{(2\pi)^{D/2} |\Sigma_m|^{1/2}} \quad (1)$$

where M is the number of mixtures in λ, D is the feature vector dimension, p_m, $\vec{\mu_m}$ and Σ_m represent the mixture weight, mean and covariance matrix of the m_{th} mixture. Usually diagonal covariances are used instead of full covariance matrix for computational and data sparity reasons.

The computation is expensive since we have to do M times exponential operations. In order to save computation load, we use log-domain approximation instead, which is:

$$log(L(x_t|\lambda)) \approx \max_{m=1}^{M} log(p_m \frac{exp\{-\frac{1}{2}(x_t-\mu_m)^{'}\Sigma_m^{-1}(x_t-\mu_m)\}}{(2\pi)^{D/2}|\Sigma_m|^{1/2}}) \quad (2)$$

In practice, in order to save model loading time and computation time, we use the following equation instead of (2):

$$log(L(\vec{x_t}|\lambda)) \approx \max_{m=1}^{M}\{C_m - \Sigma_{d=1}^{D} v_{md}(x_{td}-\mu_{md})^2\} \quad (3)$$

Where C_m is the pre-computed constant for each mixture, μ_{md} is the d_{th} dimensional mean of the m_{th} mixture, $v_{md} = 1/(2\sigma_{md}^2)$, and σ_{md}^2 is the d_{th} dimensional diagonal covariance of the m_{th} mixture.

This is the nearest-neighbor approximation framework. Under this framework, GMM computation can be regarded as finding the nearest mixture to a given feature vector.

2.2. Gaussian Box

For the m_{th} mixture, the hyper-ellipsoid region in the feature space where equation (2) gives higher likelihood scores than a given threshold T is called the Gaussian region of mixture m. The box with boundary hyperplanes orthogonal to the coordinate axes that completely includes the Gaussian region is called the Gaussian Box. In practice instead of an absolute threshold, relative threshold (by ignoring C_m) is used:

$$-v_{md}(x_{td}-\mu_{md})^2 \geq log(T) \quad (4)$$

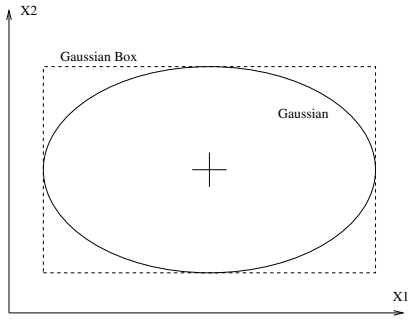

Figure 1: Gaussian Box

2.3. BBI algorithm using K-Dimensional Partitioning Tree

K-Dimensional (K-d) tree is the generalization of one dimensional binary tree. At each nonterminal node, the feature space is split into two half spaces by a hyper-plane orthogonal to one of the K coordinate axes. Each leaf defines a region which is the intersection of all the sub-spaces generated along the path from root node to the leaf node. Therefore a K-d tree partitions the feature space into several disjoint rectangular regions (called buckets) and reduces the search space from full Gaussian evaluation to only the Gaussians whose boxes intersect with the bucket containing the given feature vector.

BBI algorithm optimizes the K-d tree by minimizing the expected number of bucket-box intersections given the Gaussian boxes for an error threshold T.

3. Improvements to traditional BBI algorithm

3.1. Optimal criteria for dividing plane determination

The traditional BBI algorithm tends to build up a balanced binary tree by choosing the dividing hyper-plane as the one that has the minimum number of bucket box intersections with the restraint of having equal number of bucket box intersections at each side. However the resulting tree, although balanced, is not optimal (it is not the tree that has the minimum expected number of bucket-box intersections).

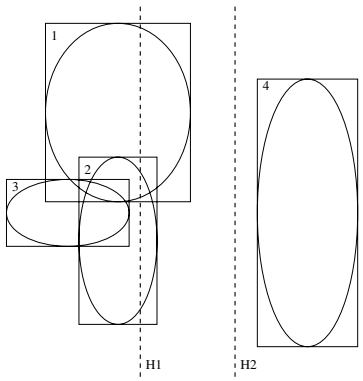

Figure 2: Example of dividing plane determination

As seen from Fig.2, the dividing plane H_2 with 3 Gaussian boxes on the left and 1 Gaussian box on the right is obviously optimal. However the traditional BBI algorithm chooses H_1 as the dividing plane because H_1 generates a balanced tree. This will results in a larger binary tree because it needs at least one more node to divide Gaussian box 4 and the other boxes.

To fix this problem, we propose to define the optimal dividing hyper-plane as the one that generates minimum expected number of mixture evaluations:

$$(K^*, H^*) = \arg\min_{k,h} P_L N_L + P_R N_R \quad (5)$$

where h is the hyper-plane on the k_{th} dimension, P_L and N_L are the probability and number of Gaussian boxes on the left side of h, P_R and N_R are the probability and number of Gaussian boxes on the right side of h. Since the given feature vector is not necessarily generated by the current Gaussian mixture model under investigation, there is no information about the a-priori probability. Therefore, without loss of generality, we can assume that each mixture is equally likely. Under this assumption, the optimal criteria can be simplified as:

$$(K^*, H^*) = \arg\min_{k,h} {N_L}^2 + {N_R}^2 \quad (6)$$

3.2. Optimal location of dividing plane

The criteria for dividing plane determination is used to find out between which two gaussian boxes we should make the cut. How-

ever the location of the cut is still unknown. The traditional BBI algorithm uses the boundaries of Gaussian boxes as the locations of dividing hyper-planes (H1 or H2 in the example shown in Fig 3. This, although simple to train, adds bias to the BBI tree and introduces potential errors to recognition. Here we propose to define the location of the dividing plane as the one that has the same Mahalanobis distance to the closest dividing mixture pair.

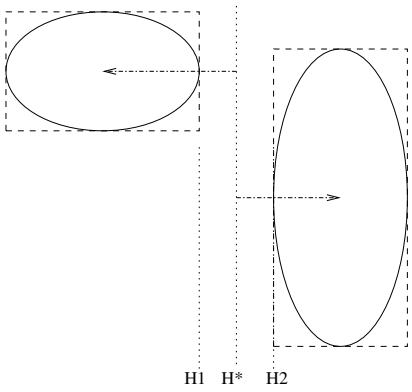

Figure 3: Optimal location of dividing plane

Suppose the optimal dividing hyper-plane lies between mixture i and mixture j in the k_{th} dimension, assuming $\mu_{ik} < \mu_{jk}$, the plane that has the same likelihoods (Mahalanobis distance) to these two mixtures can be computed from:

$$v_{ik}(H^* - \mu_{ik})^2 = v_{jk}(H^* - \mu_{jk})^2 = -log(T) \qquad (7)$$

therefore the location of optimal diving hyper-plane is,

$$H^* = \mu_{ik} + \frac{\sqrt{\frac{v_{jk}}{v_{ik}}}}{1 + \sqrt{\frac{v_{jk}}{v_{ik}}}}(\mu_{jk} - \mu_{ik}) \qquad (8)$$

where μ_{ik}, μ_{jk} represent the k_{th} dimensional mean of mixtures i, j, $v_{ik} = 1/(2\sigma_{ik}^2)$, $v_{jk} = 1/(2\sigma_{jk}^2)$, and σ_{ik}^2, σ_{jk}^2 represent the k_{th} dimensional covariance of mixtures i, j.

3.3. Cross-dimensional dividing plane

The traditional BBI algorithm only considers the hyper-planes that are orthogonal to one of the K coordinate axes as the candidates of dividing planes. Here we extend the candidates pool of dividing planes to the hyper-planes that run across two dimensions. This exponentially increases the size of candidates pool. For simplicity, currently we only consider two special cross-planes: $y = x + b$ and $y = -x + b$.

Suppose the optimal dividing hyper-plane lies between mixture 1 and mixture 2 and it runs across i_{th} and j_{th} dimension,

For $y = x + b$, without loss of generality, assuming $\mu_{1j} - \mu_{2j} > \mu_{1i} - \mu_{2i}$, the optimized location of dividing plane can be computed as:

$$B^* = \mu_{1j} - \mu_{1i} - \sqrt{\frac{v_{1i} + v_{1j}}{v_{1i} \cdot v_{1j}}} \cdot \sqrt{-logT_1^*} \qquad (9)$$

$$\text{where} \quad \sqrt{-logT_1^*} = \frac{(\mu_{1j} - \mu_{2j}) - (\mu_{1i} - \mu_{2i})}{\sqrt{\frac{v_{1i}+v_{1j}}{v_{1i} \cdot v_{1j}}} + \sqrt{\frac{v_{2i}+v_{2j}}{v_{2i} \cdot v_{2j}}}} \qquad (10)$$

For $y = -x + b$, without loss of generality, assuming $\mu_{1i} + \mu_{1j} > \mu_{2i} + \mu_{2j}$, the optimized location of dividing plane can be computed as:

$$B^* = \mu_{1j} + \mu_{1i} - \sqrt{\frac{v_{1i} + v_{1j}}{v_{1i} \cdot v_{1j}}} \cdot \sqrt{-logT_2^*} \qquad (11)$$

$$\text{where} \quad \sqrt{-logT_2^*} = \frac{(\mu_{1i} + \mu_{1j}) - (\mu_{2i} + \mu_{2j})}{\sqrt{\frac{v_{1i}+v_{1j}}{v_{1i} \cdot v_{1j}}} + \sqrt{\frac{v_{2i}+v_{2j}}{v_{2i} \cdot v_{2j}}}} \qquad (12)$$

In both cases, this restraint has to be satisfied: $-logT^* \geq -logT$. And to prevent error increase due to over-shrinkage of search space, we require each leaf of the BBI tree to have no less than 4 Gaussians attached.

4. Experiments

We evaluate the three techniques in 2 recognition domains using CASSI – the Conversay's embedded speech engine based on continuous density hidden Markov model (CDHMM). The acoustic models are state-clustered using decision-tree. We have tried both global BBI tree strategy and state-based BBI tree strategy. The latter one showed much better performance than the first one. So in the following experiments, all BBI algorithms are implemented on state-based, which means, there is one pre-computed BBI tree for each GMM. In order to restrict the size of BBI tree, the maximum allowable depth of BBI tree is set to 8. For each BBI implementation configuration, we run 7 experiments with relative threshold T ranges from 0.3 to 0.6 with step 0.05.

4.1. Aurora Digits Recognition

In this task, the acoustic model (401 senones, average 21.5 mixtures per GMM) is trained from 8116 sentences with SNR value ranges from 5 to 20 and clean. The test set consists of total 7007 sentences with SNR value ranges from -5 to 20 and clean, with 1001 sentences under each environment.

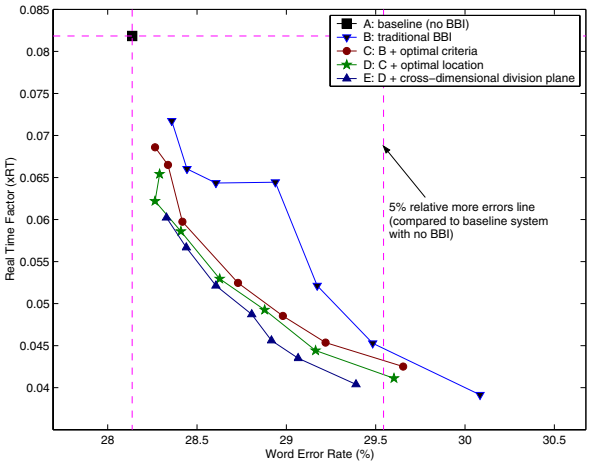

Figure 4: Word error rate V.S. recognition time for Aurora.

Fig.4 shows the curve of recognition errors V.S. running time for Aurora task. We can see from this figure that with all three improvements implemented (E), we achieve lower errors with less running time than the traditional BBI (B). This figure also shows

that compared with baseline system with no BBI (A) we can achieve 50% running time reduction with less than 5% relative error increase.

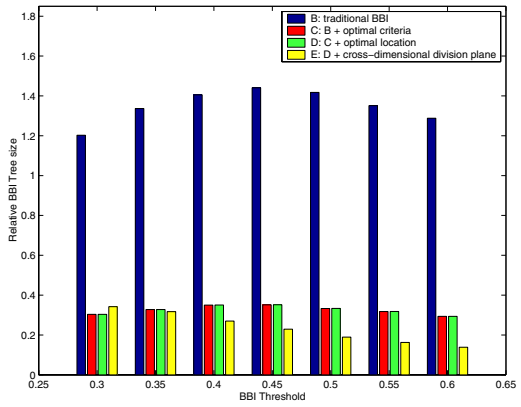

Figure 5: Relative BBI tree sizes for Aurora

Fig.5 shows the relative sizes of BBI trees for different BBI implementations. We can see that, by using optimal criteria for dividing hyper-plane determination, the size of BBI tree (C, D and E) was reduced by a large amount (only $1/4$ of the traditional BBI tree (B)), which is very important for embedded systems with limited resources.

4.2. Samsung Command-and-Control Recognition

Samsung task is a 175-word cellphone command & control recognition task. The acoustic model (2689 senones, average 17.6 mixtures per GMM) is trained from 19356 sentences. The test set has 4840 sentences.

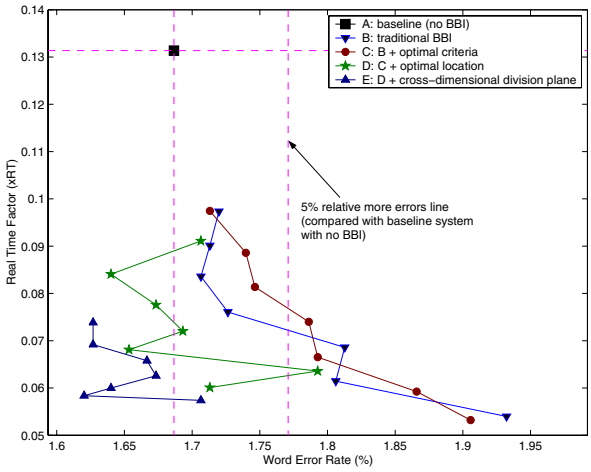

Figure 6: Word error rate V.S. recognition time for Samsung.

From Fig.6 we can see that with new BBI algorithm (E) we achieve a lower error rate with less running time than the traditional BBI (B). Compared to the baseline system with no BBI (A) we speed up the GMM computation by 50% with less than 5% relative error increase. We also notice some randomness in the graphs. The heuristic that each leaf node has at least 4 Gaussians is the probable cause for this behavior. As the search space shrinks this heuristic becomes more effective resulting in ambiguous dividing plane candidates.

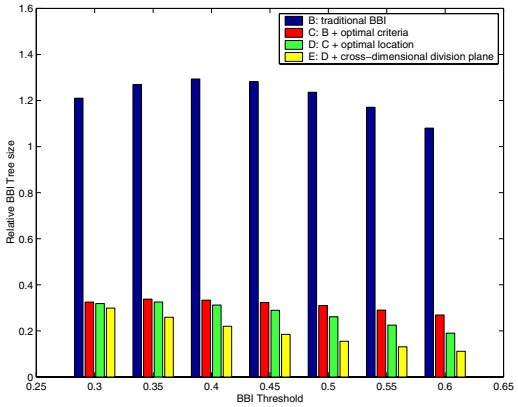

Figure 7: Relative BBI tree sizes for Samsung

Fig.7 shows the relative sizes of BBI trees for different BBI implementations. By using optimal criteria for dividing hyperplane determination, the size of new BBI tree (C, D and E) was only one fourth of the traditional BBI tree (B).

5. Conclusion

In this paper, we describe three techniques to refine the traditional BBI algorithm. Experimental results show that they give significant improvements. In the future, we will extend the cross-2-dimensional dividing plane to cross-n-dimensional dividing plane, which should be able to achieve higher speedup.

6. References

[1] A. Chan, J. Sherwan, R. Mosur, and A. I. Rudnicky, "Four-level categorization scheme of fast GMM computation techniques in large vocabulary continuous speech recognition systems," in *IEEE ICSLP'04*, 2004.

[2] Bryan L. Pellom, Ruhi Sarikaya, and John H. L. Hansen, "Fast likelihood computation in nearest-neighbor based search for-continuous speech recognition," *IEEE Signal Processing Letters*, vol. 8, no. 8, pp. 221–224, 2001.

[3] A. Lee, T. Kawahara, and K. Shikano, "Gaussian mixture selection using context-independent HMM," in *IEEE ICASSP'01*, 2001, vol. 1, pp. 69–72.

[4] M.J.F. Gales, K.M. Knill, and S.J. Young, "State-based gaussian selection in large vocabulary continuous speech recognition using HMMs," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 2, pp. 152–161, 1999.

[5] J. Fritsch and I. Rogina, "The Bucket Box Intersection (BBI) algorithm for fast approximative evaluation of diagonal mixture gaussians," in *IEEE ICASSP'96*, 1996, vol. 2, pp. 837–840.

[6] V. Ramasubramanian and Kuldip K. Paliwal, "Fast k-dimensional tree algorithms for nearest neighbor search with application to vector quantization coding," *IEEE Transactions on Signal Processing*, vol. 40, no. 3, pp. 518–531, 1992.