Fast Gaussian Evaluations in Large Vocabulary Continuous Speech Recognition

October 25th, 2002

Shivali Srivastava

Candidate for Master of Science in Electrical Engineering
Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University

Organization of Presentation

* Motivation for fast Gaussian computation (FGC)
* Bucket Box Intersection (BBI) algorithm and issues related to the BBI algorithm
* Extended BBI (EBBI) algorithm
* Comparison between the EBBI and BBI algorithms
* Comparison between the EBBI algorithm and Gaussian Clipping (GC) technique
* Experimental results and analysis
* Conclusions and future work

Motivation

* State likelihood computations in the statistical modeling component of the speech recognition systems contribute 50%-80% to the computational load

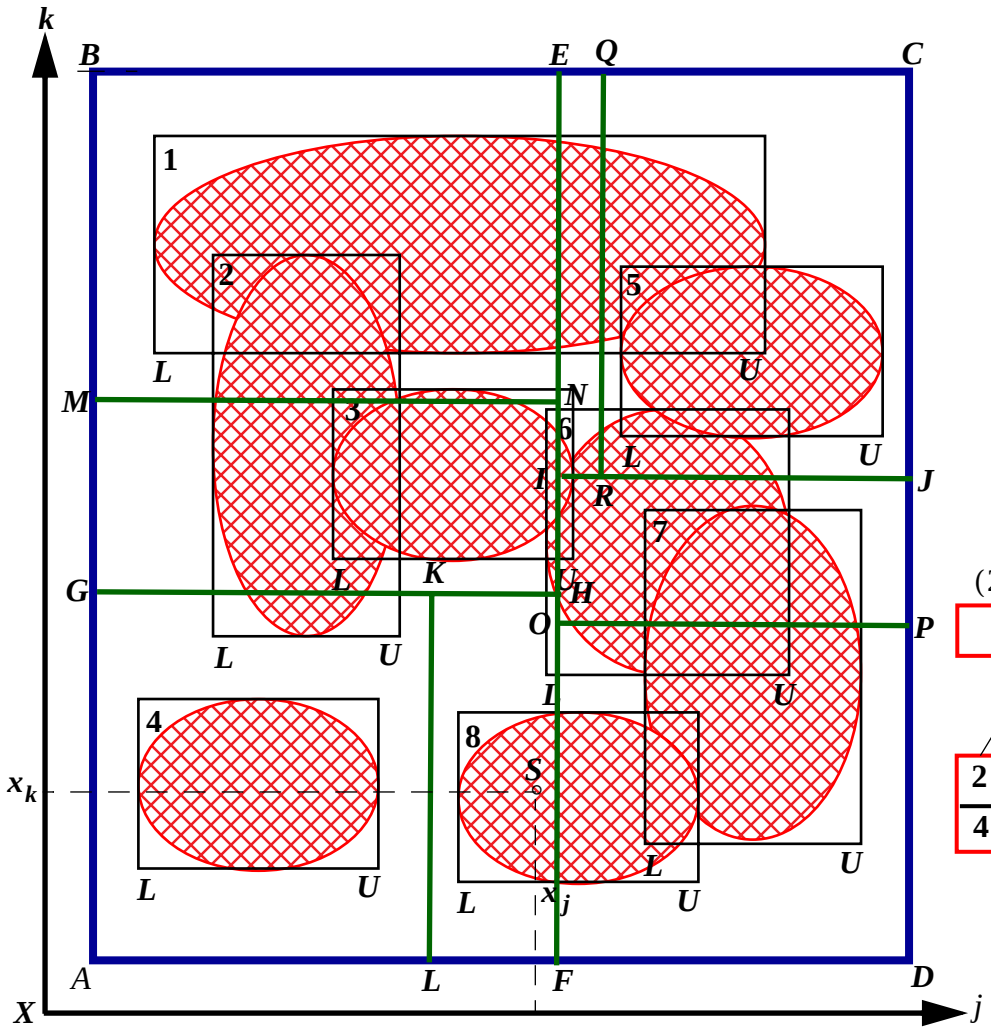
* The likelihood computation is dominated by a few (significant) Gaussian components and can be computed solely from these Gaussian components without a significant increase in the system WER

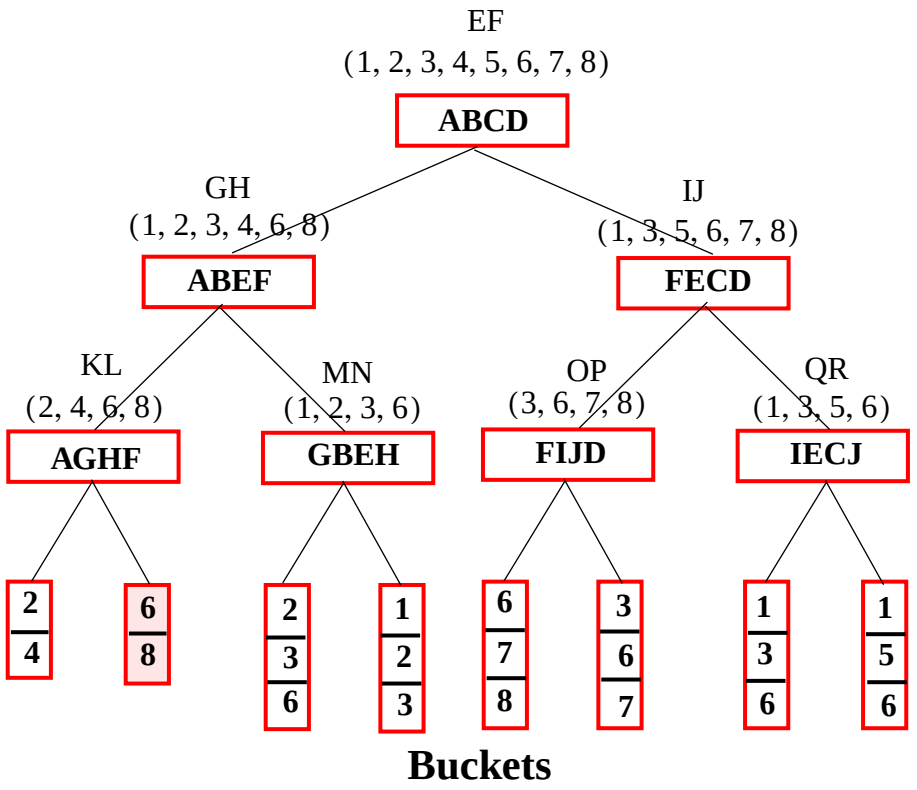
* Need an efficient way to find the most significant Gaussians for the likelihood computation

BBI Algorithm (I)

* Use of a pre-computed multi-dimensional decision tree (k-d tree) to determine the set of the most significant Gaussians
* Represents all the Gaussians in the mixture by using the Gaussian boxes and uses the Gaussian boxes to build the k-d trees
* Restrict the likelihood computation to the Gaussians with boxes that contain the vector
* Algorithm parameters — tree depth and relative threshold

BBI Algorithm (II)





Issues Related to the BBI Algorithm

Optimization criterion:

- Optimization criterion used by the BBI algorithm requires the selection of a balanced hyperplane that produces a minimum number of Gaussian splits

- In practical applications there may be more than one collection of hyperplanes that produce the same (minimum) number of Gaussian splits

Mixture weight re-normalization:

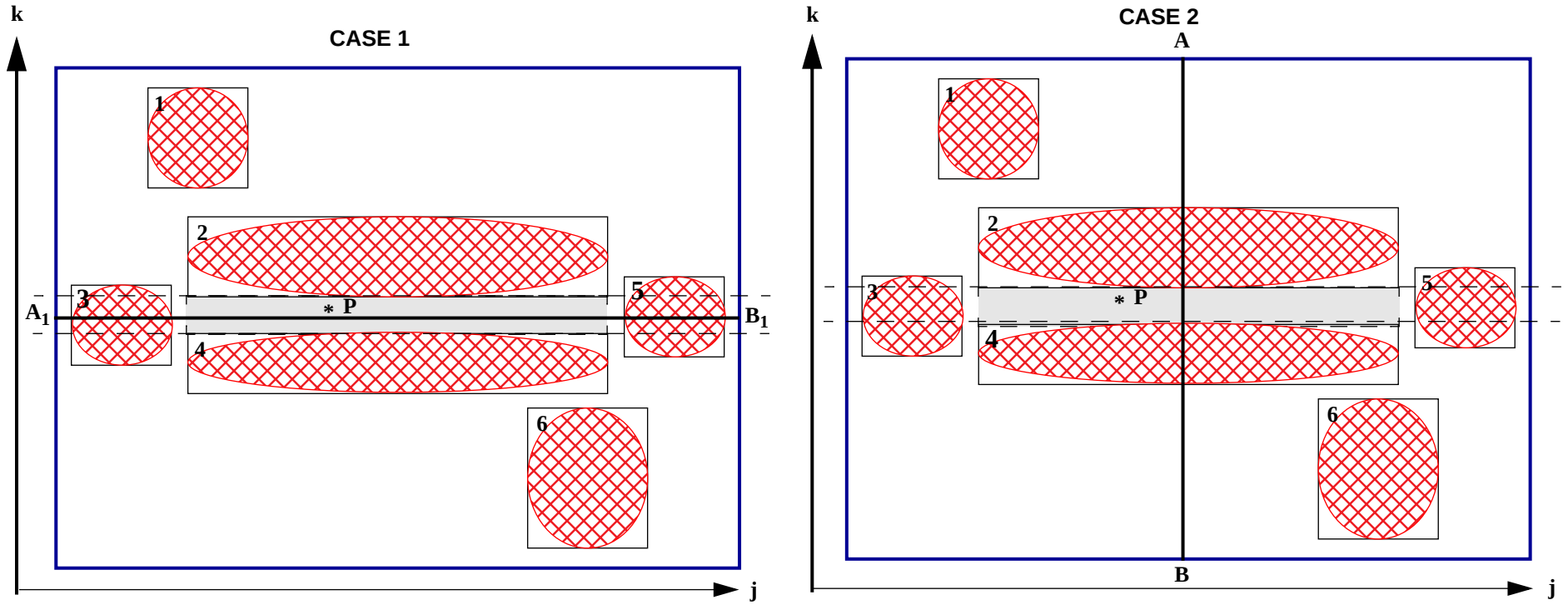
- BBI algorithm restricts the likelihood computation to the Gaussians that contain the feature vector but it doesn't re-normalize the mixture weights

EBBI Algorithm

A Modified Optimization Criterion (I)

* The modified optimization criterion for the multiple minima in Gaussian splits includes following steps:
  * Finding the variance of those coordinate axes whose hyperplanes produce a minimum number of Gaussian splits
  * Choosing the hyperplane corresponding to the coordinate axis with the highest variance

A Modified Optimization Criterion (II)



* Given the positions and forms of the Gaussians in the region, Gaussians 2 and 4 can contain the vector *P* (shown in the figure) with an equal probability

A Modified Optimization Criterion (III)

* Hyperplanes AB and A_1B_1 give a balanced division and produce the same (= 2) number of Gaussian splits
* A k-d tree obtained by using the hyperplane A_1B_1 produces a higher approximation error
* Under the framework of the division hyperplane AB, Gaussians 2 and 4 both contain the vector P
* A k-d tree obtained by using the hyperplane AB produces a lower approximation error
* In general, the hyperplane AB produces a lower approximation error for all the vectors in the shaded region

Mixture Weight Re-normalization (I)

* The log-likelihood of a feature vector in a k-dimensional space is $L = \sum_{m=1}^{M} \log(w_m \cdot P_m)$, where, w_m is the weight of the m^{th} Gaussian and P_m is the pdf of the m^{th} Gaussian in a mixture of M Gaussians

* If a feature vector lies in a bucket containing the Gaussians l and $l + p$, the log-likelihood of the vector is: $L = \log(w_l \cdot P_l) + \log(w_{l+p} \cdot P_{l+p})$

Mixture Weight Re-normalization (II)

* Using normalized weights, the log-likelihood is:

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

* If $w_l + w_{l+p} = C$, $L' = L - 2 \cdot \log C$

* Since $C \leq 1$ or, $\log C \leq 0$, $L' \geq L$

* Use of re-normalized weights results in higher log-likelihoods, which in turn leads to an improved system performance

Selection of a K-d Tree

* There are several options for how we apply a k-d tree to the HMM model (i) one k-d tree per state, (ii)) a single large k-d tree shared by all the HMM states, or (iii)multiple k-d trees with each tree sharing a subset of the states

* Previous implementations of the BBI algorithm successfully employed the latter two options

* The EBBI algorithm uses one k-d tree for each HMM state, because it results in the lowest approximation error and produces the significant speedups as compared to the other two options

EBBI and BBI Algorithms (I)

* Industry-standard OGI-Alphadigits database (telephone database of 6-word strings) used
* 39-dimensional MFCC features used
* 64-mixture cross-word triphone models used
* K-d trees with a tree depth 6 and an error threshold 0.4 used for the two algorithms
* The baseline system produced a 10.1% WER and used 57.9% of the total CPU time for the Gaussian Computations

EBBI and BBI Algorithms (II)

Algorithm	% of Total CPU Time used by Gaussian Evaluations	WER (%)
EBBI	29.1	10.2
BBI	24.7	11.4

* The EBBI algorithm used the modified optimization criterion, mixture weight re-normalization and state level k-d trees
* The EBBI algorithm produced a significant improvement in WER over the BBI algorithm
* The EBBI algorithm produced only a 1.8% lower speedup than the BBI algorithm (not a significant difference)

EBBI Algorithm and GC Technique

* Gaussian clipping technique is used to generate lower-order mixture models from higher-order mixture models by removing the Gaussians with the lower mixture weights

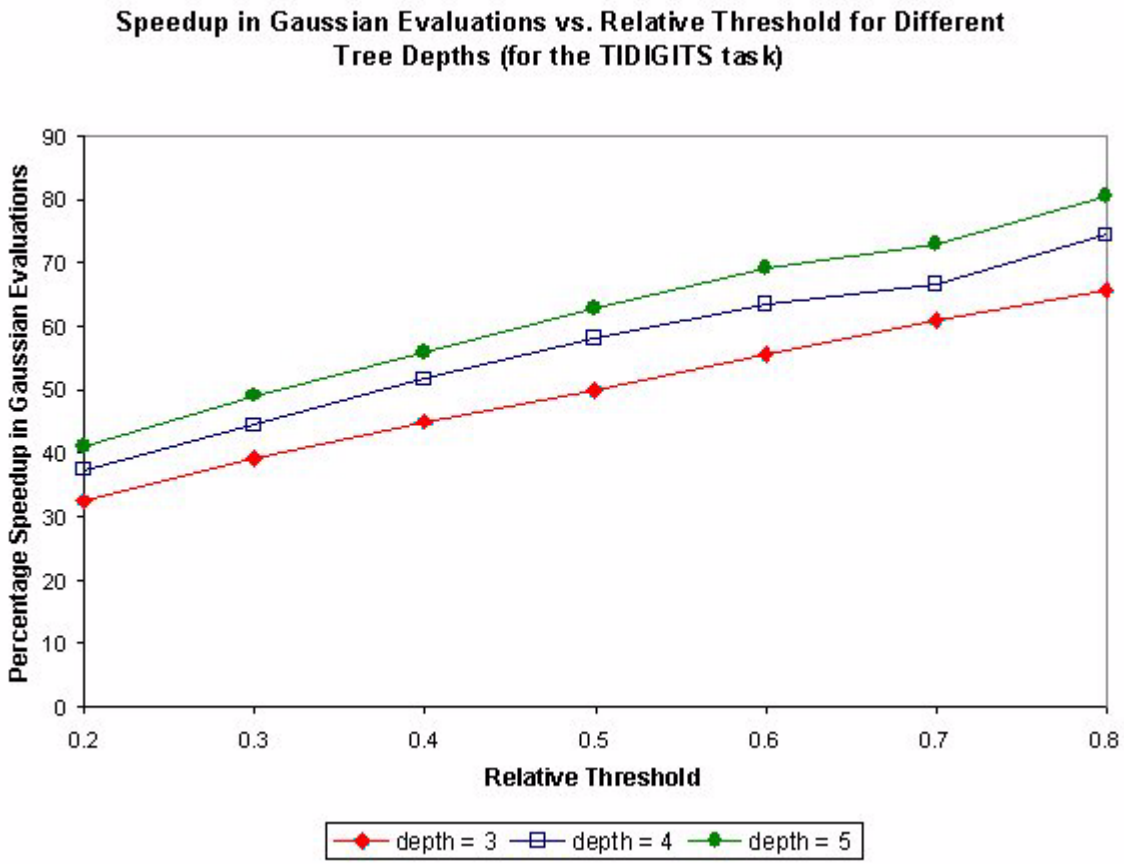
Gaussian speedup (%)	Technique used	WER (%)
25	EBBI	10.3
	GC	11.5
50	EBBI	10.7
	GC	12.2

* For a 50% speedup in the likelihood computation, the EBBI algorithm produced a 12.3% improvement over the GC approach

Performance of the EBBI Algorithm
TIDIGITS Database

* A small database containing continuous digits (vocabulary size of 11)
* 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
* The system which is not using the EBBI algorithm (a No EBBI system) produced a 0.6% WER
* Gaussian evaluations took 24.4% of the total CPU time in this system

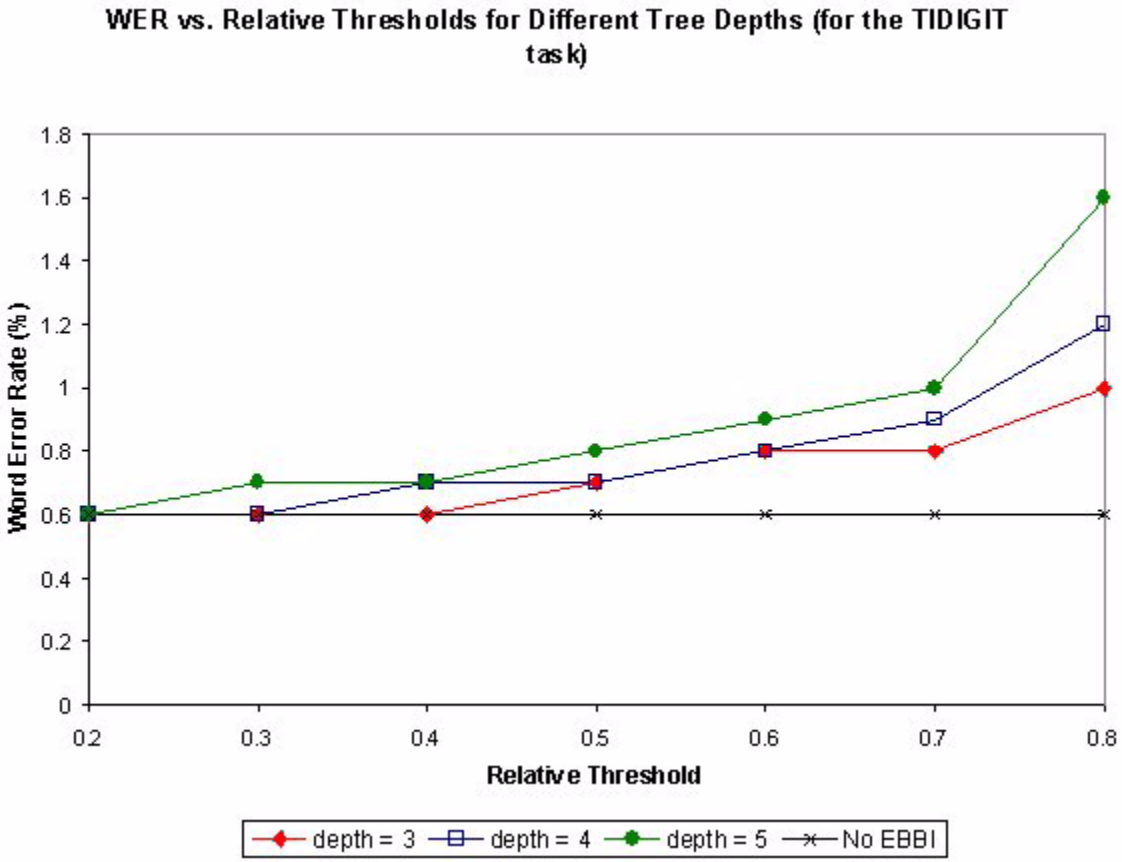
TIDIGITS - Gaussian Speedup



* For a fixed tree depth, the Gaussian speedup increases with an increase in the relative threshold

* For a fixed threshold, the Gaussian speedup increases with an increase in the k-d tree depth

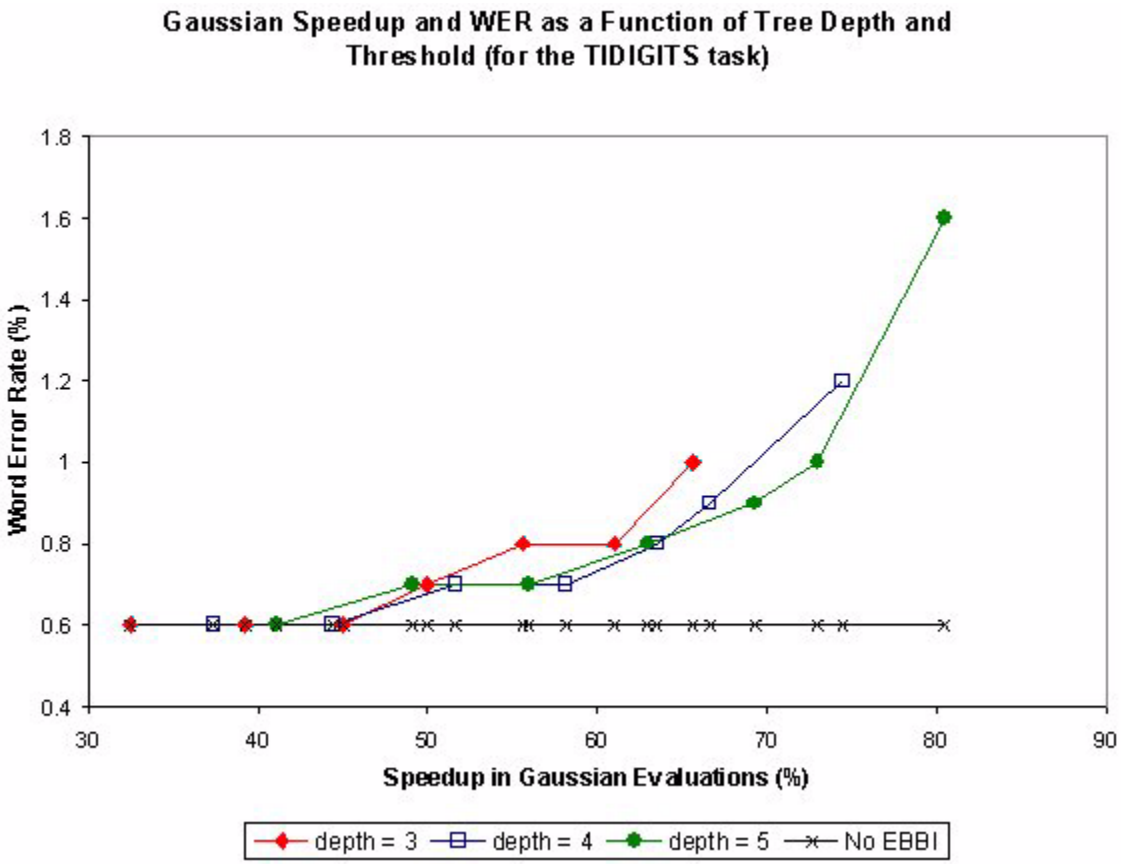
TIDIGITS - WER



* For a fixed tree depth, the WER increases with an increase in the relative threshold

* For a fixed threshold, the WER increases with an increase in the k-d tree depth

TIDIGITS - Speedup and WER



* Best performance — a tree depth of 3 with a relative threshold of 0.4 produced a 45% speedup in score computations without degrading the performance of the system

OGI-AD Database

* Telephone database of 6-word strings
* 39-dimensional MFCC features used
* HMM system used cross-word context-dependent triphone models
* 5-state left-to-right models used
* Used to evaluate the algorithm performance as the function of the number of mixture components — 16, 32 and 64 mixture Gaussians per state used

AD - 16 Mixture Components (I)

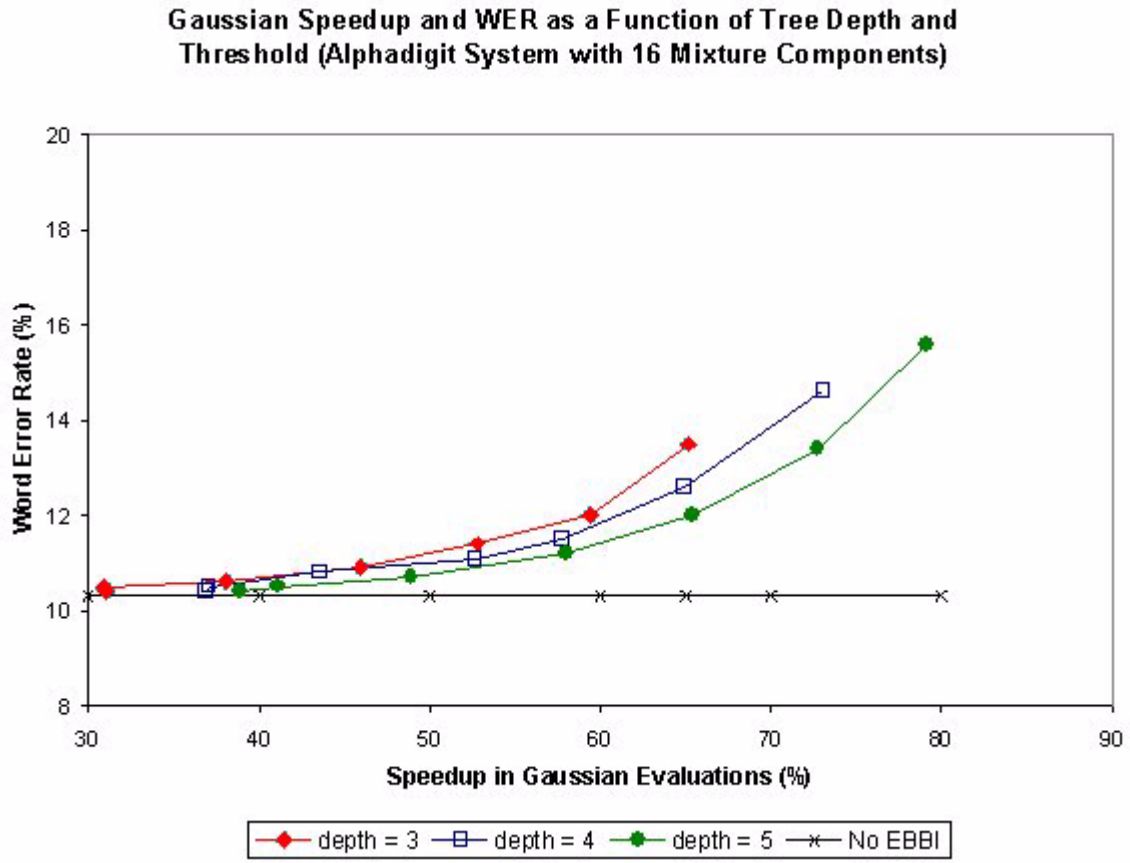
* The relative threshold was varied from 0.2 to 0.8 and the k-d tree depth was varied from 3 to 5

* System that didn't use the EBBI algorithm produced a 10.3% WER and it used 31.7% of the total CPU time for the Gaussian evaluations

* For a depth of 5, as the threshold increased from 0.2 to 0.8, the Gaussian speedup increased from 39% to 79% and the WER increased from 10.4% to 15.6%

* For a relative threshold of 0.5, as the tree depth increased from 3 to 5, the Gaussian speedup increased from 46% to 58% and the WER increased from 10.9% to 11.2%

AD - 16 Mixture Components (II)



* Best performance — a tree depth of 5 and a threshold of 0.3 produced a 41% speedup with only a 2% relative degradation in WER

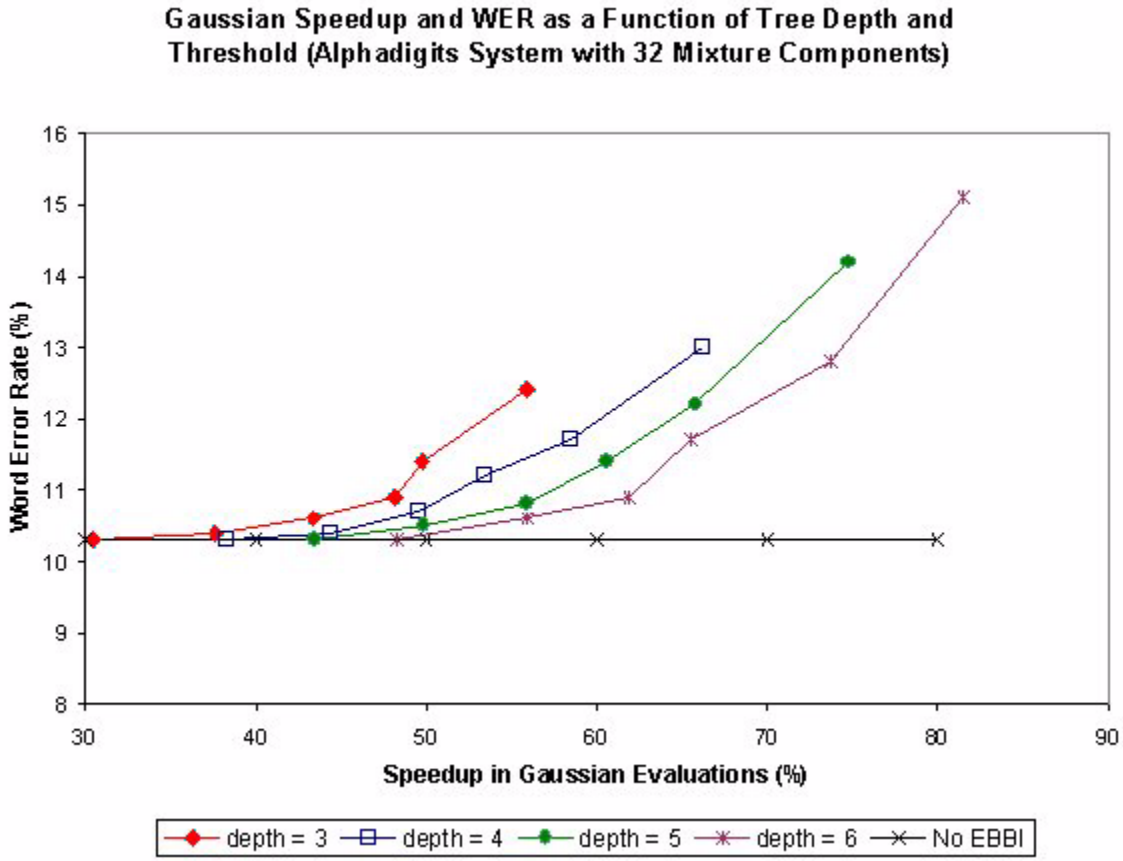
AD - 32 Mixture Components (I)

* The relative threshold was varied from 0.3 to 0.8 and the k-d tree depth was varied from 3 to 6

* System that didn’t use the EBBI algorithm produced a 10.3% WER and it used 43.1% of the total CPU time for the Gaussian evaluations

* The CPU time used by the Gaussian evaluations is 36% higher than the CPU time used by the Gaussian evaluations in the system using 16-mixture components

AD - 32 Mixture Components (II)



* Best performance — a tree depth of 6 and a threshold of 0.3 produced a 48% speedup without degrading the performance of the system

Effect of Mixture Components

* The algorithm becomes more attractive as the number of Gaussians increases

* A 48% speedup is obtained,

  * without any approximation error when 32-mixture components were used

  * with a 4% relative approximation error when 16-mixture components were used

* In general, the algorithm produces the same amount of speedup with a significantly lower increase in WER for the systems with higher mixture components

SWB Database

* HMM system used cross-word context-dependent triphone models
* 39-dimensional MFCC features used
* The relative threshold was varied from 0.3 to 0.8 and the k-d tree depth was varied from 3 to 5
* System that didn’t use the EBBI algorithm produced a 41.1% WER and it used 9.9% of the total CPU time for the Gaussian evaluations
* The EBBI algorithm doubled the speed of the score computation with only a 3.4% relative increase in the WER

Conclusions

* Compared to the BBI algorithm, the EBBI algorithm produced a significantly lower WER (11.4% vs. 10.2%) with only a 1.8% reduction in speedup

* Compared to the GC technique, the EBBI algorithm produced a significantly lower WER (12.2% vs. 10.7%) for a 50% speedup in the likelihood computation

* Algorithm performance for various databases
  * TIDIGITS — a 45% speedup without any degradation in the system performance
  * AD — a 48% speedup without any degradation in the system performance
  * SWB — a 50% speedup with only a 3.4% relative increase in the system WER

Future Work

* Need to study the performance of the algorithm under various possible hyperplanes resulting in a balanced distribution for a coordinate axis
  * the k-d trees obtained by using different hyperplanes may result in a different set of most significant Gaussians
* Include the mixture weights in computing the Gaussian boxes
  * may improve the algorithm performance without an additional computational cost during recognition

References

1. J. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol. 18, no. 5, pp. 509-517, September 1975.

2. 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.

3. 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.

4. 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.