Pak. J. Engg. & Appl. Sci. Vol. 11, July, 2012 (*p.* 1-13)

Toggling and Circular Partial Distortion Elimination Algorithms to Speedup Speaker Identification based on Vector Quantization

Muhammad Afzal[1], Mohammad A. Maud[1], and Ali Hammad Akbar[1]
shmafzal@yahoo.com mamaud@uet.edu.pk ahakbar@gmail.com

1. Department of Computer Science and Engineering, University of Engineering and Technology, Lahore, Pakistan.

Abstract

Vector quantization (VQ) efficiently competes with contemporary speaker identification techniques. However, VQ-based real-time speaker identification systems suffer latency due to distance computation between a large number of feature vectors and code vectors of speakers' codebooks to find the best match in the database. The identification time depends on dimension and count of extracted feature vectors as well as the number of codebooks. Previous speedup techniques in VQ-based speaker identification decrease test vector count through pre-quantization and prune out unlikely speakers. However reported speedup factors come with accuracy degradation. This paper proposes techniques to speedup closest code vector search (CCS) based on stationarity of speech. In this paper proximity relationship is substantiated among code vectors extracted through LBG process of codebook generation. Based upon the high correlation of proximate code vectors, circular partial distortion elimination (CPDE) and toggling-CPDE algorithms have been proposed in this paper to speedup CCS. Further speedup is proposed through pruning test feature vector sequence for unlikely codebooks during best match speaker search. Our empirical results show that an average speedup factor up to 5.8 for 630 registered speakers of TIMIT 8kHz corpus and 6.6 for 230 speakers of NIST-1999 database have been achieved through integrating the proposed techniques.

Key Words: Speaker identification; distortion computation; partial distortion elimination; vector quantization; speech stationarity utilization; closest code vector search

1. Introduction

Automated speaker identification (ASI) is defined as identifying a person based on her or his speech against speaker models [1]. These speaker models are prepared from a sequence of feature vectors extracted through processing available speech samples of persons at registration time. A similarity measure is then calculated while comparing the sequence of feature vectors X extracted from a test speech sample with models of registered speakers. An ASI system finds the best matching model amongst the registered or known speakers [2].

Most multi-user speech processing systems employ ASI front end to adapt to the current user to deliver better user specific services. Glaser and Bimbot [3] used speaker identification for verification tasks in real-world telecom applications. A fast real-time ASI system trained for wanted persons can be used to track their appearance on digital telephone networks. Real time speaker identification requirement has increased emphasis on speeding up feature matching techniques. Since Vector quantization (VQ) often outperforms GMM [2] in ASI in terms of accuracy and speed, this paper focuses on VQ.

Mel-frequency cepstrum coefficients (MFCC) are commonly used in ASI systems [4]. VQ based speaker model training maps the sequence of feature vectors, $\tilde{X} = \{\tilde{x}_{\tilde{i}} \,|\, 1 \le \tilde{i} \le \tilde{T}, \tilde{x}_{\tilde{i}} \in \mathbb{R}^d\}$, extracted from available speech samples of a speaker, to a set of M centroids or code vectors called codebook, $C = \{c_m \,|\, 1 \le m \le M, c_m \in \mathbb{R}^d\}$ through identifying M clusters of similar vectors in $\tilde{X}$ such that $M << \tilde{T}$. Linde Buzo Gray (LBG)

clustering algorithm is mostly used [5] to compute VQ codebooks that are stored in a database of registered speakers.

Consider a sequence of feature vectors $X = \{x_i \,|\, 1 \le i \le T, x_i \in \mathbb{R}^d\}$, extracted from test speech samples of a person. During pattern matching an ASI system computes quantization distortion of X with codebook C of each registered speaker according to Equation (1), wherein the test speaker is identified as the registered speaker whose codebook has minimum distortion [1].

$$D(X, C) = \sum_{i=1}^{T} e(x_i, C) \tag{1}$$

Where $e(x_i, C) = \min_{c_m \in C \,|\, 1 \le m \le M} \|x_i - c_m\|$ defines distortion of a test vector, x_i, with a codebook, C, based on Euclidean distance (EUD). The computation of $e(x_i, C)$ is regarded as closest code vector search (CCS). According to Equation (1) the time order complexity of computing minimum total distortion of X with M sized codebooks of N registered speakers is given as $O(dMNT)$ [4]. It specifically requires $2 \times d \times M \times N \times T$ floating point additions and $d \times M \times N \times T$ floating point multiplications to complete $N \times T$ number of CCSs. The research-based efforts on speeding up ASI systems attempt to expedite CCS computation on application specific aspects of VQ-based systems.

Kinnunen et al., [4] used vantage point tree (VPT) indexing to speedup CCS. They also combined VPT with pre-quantization of X and heuristic pruning of unlikely speakers to speedup up ASI. However, their reported speedup factors come with accuracy loss. How much pre-quantization and speaker pruning can be done without accuracy degradation is an unsolved question. Accuracy degradation, shown in [4] due to feature distortion by pre-quantization and fallibility of heuristics, underpin the need of faster but accurate techniques. This paper focuses on speeding up ASI while avoiding accuracy degradation. The CCS that plays crucial role in ASI speed performance has been improved in the light of a novel insight into LBG codebook generation process. Contrary to Paliwal and Ramasubramanian [9], circular partial distortion elimination (CPDE) and its faster variant TCPDE algorithms proposed in this paper have been deduced from our substantiation of proximity in code vectors of LBG-generated codebook as such. The performance of proposed algorithms has been analyzed both in terms of execution time and number of MACs (multiplications, additions, and comparisons) saved with respect to baseline systems. The rest of the paper is organized as follows: Section 2 discusses previous work on speeding up CCS and ASI. Section 3 describes the proposed speedup framework consisting of CPDE, TCPDE and VSP algorithms. The experimental parameters are described in section 4 along with discussions on results of proposed techniques. Conclusions are drawn in section 5. More detail about speech data selection and feature vector extraction is included in an appendix.

2. Related Work

The experimental study of Kinnunen et al., [4] is one of the comprehensive works which is frequently referenced for speaker identification speedup. The VPT indexing employed in [4] to speedup CCS in their ASI systems resulted in 24% speedup for codebook of size 256. VPT is a balanced binary tree of code vectors which in the best case takes $O(\log_2 M)$ EUD computations to complete a CCS [4]. Their paper also studied feature vector pre-quantization to reduce T which tends to distort the speaker specific characteristics ingrained in X. Further speedup studies of [4] are based on heuristic pruning of unlikely codebooks. In all, Kinnunen et al., [4] studied three speed controlling parameters namely, M, N and T by combining afore mentioned speedup techniques. The parameter d left unconsidered in [4] has been studied in [7] for speeding up ASI through partial distortion elimination (PDE) proposed in [5] and [8].

PDE speedups CCS by terminating computation of d dimensional EUD distance, as

the sum $\sum_{j=1}^{d}(x_i[j]-c_m[j])^2$ becomes equal or greater than currently minimum squared distance at $j < d$. Ramasubramanian and Paliwal [6] have reclassified previously proposed techniques to speedup CCS under approximation and elimination frame work. Most of these techniques focus on reducing the effect of the parameter M by explicitly approximating the closest code vector and eliminating unlikely ones. These techniques suffer from high overhead of approximation and elimination which increases with dimensionality [9].

An improvement is proposed in [9] for PDE (to be called CSPDE in this paper) by reordering of code vectors of LBG generated codebook in decreasing order of clusters size. CSPDE [9] used 20 seconds long raw speech sampled at 8 kHz to obtain vectors of dimension d = 4, 5, 6, 7, 8, 9, 10 by shifting 1 value for the next vector. The multiplication operations saved for codebooks sizes $\left\{2^d \times d \mid 4 \le d \le 10, d \in \mathbb{N}\right\}$ were {32.7%, 20.9%, 19%, 20.8%, 10.6%, 10.2%, 4.3%} [9] respectively, compared with the plain PDE. The results however, showed a decrease in elimination efficiency of CSPDE as compared to PDE for larger vectors and larger codebooks. The static and implicit approximation of CSPDE that has resulted in small elimination for large vectors is due to high entropy of code vectors [6, 9]. Since ASI systems calculate $D(X,C)$ for all registered codebook to find the best match so sorted code vectors of other $N-1$ codebooks might not be favorable for speeding up CCS on the whole.

The overall problem addressed in this paper is speeding up VQ based ASI without accuracy loss. The stationarity of speech has largely been unfocussed in previous studies for faster CCS. We propose algorithms that track the variation of speech in X by altering the scan order of code vectors, taking advantage of proximity of code vectors to hit the closest code vector the earliest. Our algorithms depart from Voronoi view of codebooks that depict random placement of code vectors in $\mathbb{R}^d$ space and highlight proximity relationship of code vectors. In this paper speed controlling parameter d and T of ASI systems are empirically investigated with more emphasis on previously less emphasized parameter d.

3. The Proposed Techniques

It is concluded in [9] that LBG-generated codebooks have no favorable order for speeding up PDE. This conclusion is misconstrued and insufficient. Voronoi view [13] of LBG-generated codebook, as shown in Figure 1(a), does not show any proximity relationship among the code vectors. In this paper, we present view of an intrinsic structure of LBG-generated codebook that is natural to the construction process of LBG algorithm. LBG process of codebook generation provides us information that can be utilized for efficient scan ordering.

LBG progressively generates double sized codebook from the lastly generated codebook starting from codebook of size 1. This is done by splitting each cluster of training vectors in previous codebook into a pair of smaller clusters followed by tuning of new clusters. Each cluster splitting step creates one new code vector away from the previous code vector in one direction of the d axis and another one in correspondingly opposite direction of the *d* axis of the code vector space $\mathbb{R}^d$. The tuning step of LBG iterates through the training vectors, $\tilde{X}$, till distortion fails to improve. Each iteration checks membership of the training vectors in the new clusters, updates the new code vector based on the latest membership and the computes the recent total distortion. This progress in codebook development through LBG in terms of relationship between indexes of code vectors in latest codebook and the previous codebook is shown in Figure 1(b).

Therefore, proximity of code vectors in an LBG-generated codebook follows a trend. The code vectors occurring more adjacent in the codebook tend to be more similar than the ones falling farther apart. Three dimensional plot of Figure 2 depicts the distance data averaged from codebooks of 230 speakers of NIST 1999 speech data. The distances between every pair of code-vectors of 230 codebooks were computed for $32 \times 32 = 1024$ pairs of indexes for each codebook of size 32×15. An average distance of 230 codebooks was calculated for each pair of index. The symmetry in Figure 2 shows an overall

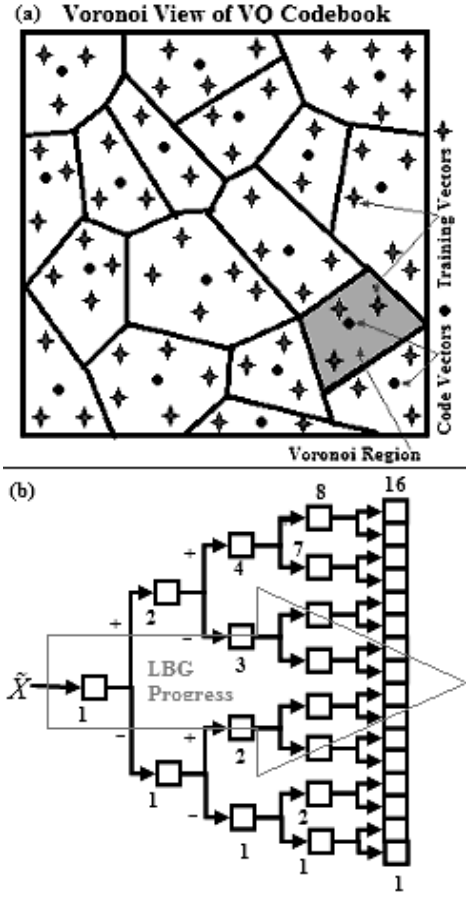

Fig.1 LBG Codebook: (a) Typical Voronoi view (b) LBG codebook generation, more similar code vectors placed at more adjacent indexes

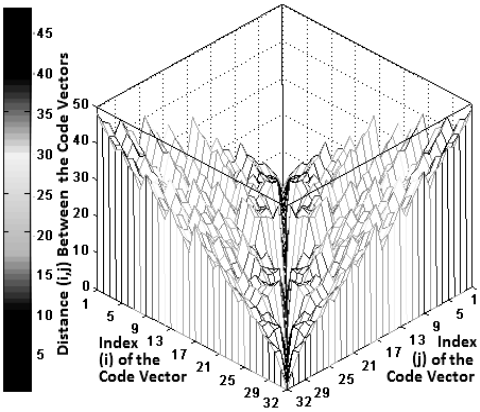

Fig. 2 Proximity plot of code vectors in terms of distance between pairs of code vectors of 32 sized codebooks of NIST-1999 data.

trend of decrease in distance between code vectors if difference between their indexes in less. This proximity relationship among code vectors sets bases of multiple favorable routes for CCS as capitalized in the proposed algorithms of this paper.

To efficiently utilize proximity of adjacent code vectors and stationarity of *X*, we propose CPDE (Algorithm 1) for making CCS fast. CPDE starts each CCS at code vector index vi for every test vector using heuristic approximation based on stationarity of *X*. For $x_i \mid 2 \le i \le T$ value of vi must be the index of closest code vector found in previous CCS made for x_{i-1}. For x_1, value of vi may be set $1 \le vi \le M$ by the calling Algorithm 3. The test feature vectors and codebooks are inputs for CPDE.

Algorithm 1: Computing vector distortion with CPDE

1: Set $m = vi\,; \sigma' = \infty\,; k = 1$

2: Do

2.1: Set $j = 1\,; \sigma = 0$

2.2: Do

2.2.1: Set $\sigma = \sigma + (x_i[j] - c_m[j])^2$

2.2.2: *if* $\sigma \ge \sigma'$ *goto* 2.4:

2.2.3: Set $j = j + 1$

while $j \le d$

2.3: Set $\sigma' = \sigma; vi = m$

▼ Select next code vector or the first one

▼ if last was scanned currently

2.4: Set $m = m + 1;$ *if* $m > M$ *then* $m = 1$

2.5: Set $k = k + 1;$

while $k \le M$

3: Set $e(x_i, C) = \sqrt{\sigma'}$

CPDE can be reduced to PDE algorithm by eliminating step 2.4 as well as the two assignments involving vi and then replacing k by m in the algorithm. CSPDE is in fact PDE with rearranged codebooks. CPDE locates the closest code vector earlier than PDE and sets σ' to the lowest value during CCS that causes elimination condition, $\sigma \ge \sigma'$, to occur at smaller values of j for remaining code vectors. CPDE

follows a unidirectional circular scan of codebook. In some cases, CPDE delays testing of the closest code vector which is lying on the opposite side of the circular scan.

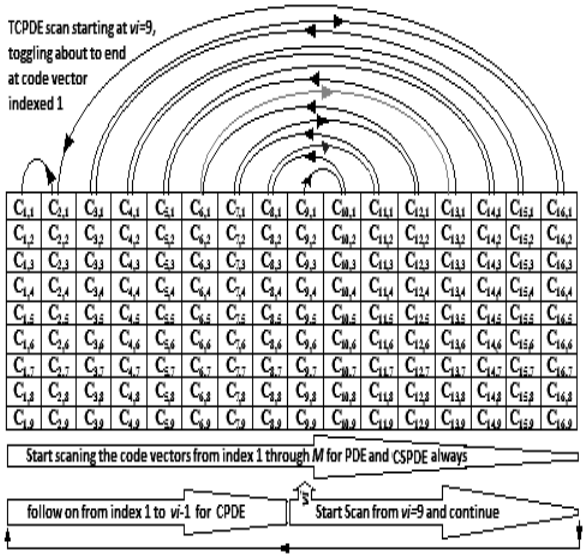

Fig. 3 Typical search paths for PDE, CSPDE, CPDE and TCPDE for vi = 9.

To reduce the likelihood of this delay, CPDE is further improved by checking next adjacent code vectors through switching back and forth between clockwise and anti-clockwise directions. TCPDE locates the closest code vector earlier more frequently than CPDE and causes comparatively higher level of elimination to speedup CCS further. Figure 3 shows typical search paths for various PDEs studied in this paper. Algorithm 2 describes TCPDE which toggles during CCS.

Algorithm 2: Computing vector distortion with TCPDE

1: Set $m = vi$; $m' = vi$; $\sigma' = \infty$; $k = 1$

2: Do

2.1: Set $j = 1$; $\sigma = 0$

▼ Clockwise scan of codebook

2.2: Do

2.2.1: Set $\sigma = \sigma + (x_i[j] - c_m[j])^2$

2.2.2 *If* $\sigma = \sigma + (x_i[j] - c_m[j])^2$

2.2.3 Set $j = j + 1$

while $j \le d$

2.3: Set $\sigma' = \sigma$; $vi = m$

▼ For clockwise scan select previous code vector

▼ or the last one if 1st was scanned currently

2.4: Set $m = m - 1$; *if* $m < 1$ *then* $m = M$

▼ For anti-clockwise scan select previous code

▼ vector or the last one if 1st was scanned currently

2.5: Set $m' = m' + 1$; *if* $m' > M$ *then* $m' = 1$

2.6 Set $j = 1$; $\sigma = 0$

▼ Anti clockwise scan of codebook

2.7 Do

2.7.1: Set $\sigma = \sigma + (x_i[j] - c_{m'}[j])^2$

2.7.2: *if* $\sigma \ge \sigma'$ *goto* 2.9:

2.7.3: Set $j = j + 1$

while $j \le d$

2.8 Set $\sigma' = \sigma$; $vi = m'$

▼ Advancing scan in both directions

2.9 Set $k = k + 1$

while $2 \times k \le M$

3: Set $e(x_i, C) = \sqrt{\sigma'}$

Regarding floating point operations, the consecutive steps '$\sigma = \sigma + (x_i[j] - c_{m'}[j])^2$' and '*if* $\sigma \ge \sigma'$ *goto label* :' (to be called **core-steps**) are common in PDE, CSPDE, CPDE and TCPDE. The core-steps involve (1, 2, 1) MACs of floating point numbers. A full CCS that computes all distances completely performs first core-step $d \times M$ times without performing second step but computes $\sigma \ge \sigma'$ condition M times and hence involves $M \times (d, 2d, 1)$ MACs. All variants of PDE perform the core-steps $d' \times M$ times and hence involve $M \times d' \times (1, 2, 1)$ MACs, where d' is average of j values at which $\sigma \ge \sigma'$ condition occurs for M code vectors. Table 1 shows the worst, best and average case analysis with underlying assumptions for all variants of PDE.

Table 1: FLOPS analysis for all variants of PDE for best, worst and average case

Case	Assuming $\sigma \geq \sigma'$ becomes true	MACs $\times(1,2,1)$
Worst	at $j = d$ or never $\forall m \mid 1 \leq m \leq M$	$M \times d$
Best	at $j = 1$ $\forall m \mid 2 \leq m \leq M$	$M + d - 1$
Average	(worst + best) /2	$\frac{Md + M + d - 1}{2}$

Regarding integer operations, CPDE algorithm performs *M* additions and comparisons more than PDE, whereas TCPDE algorithm performs *M/2* integer additions and comparisons more than CPDE.

All variants of PDE as well as full CCS require *d*×*M*×*N* and *d*×*T* floating point storage location to store registered code books and test feature vector sequence, respectively. Both CPDE and TCPDE require only a single extra storage to store the index of code vector best matched with the previous test vector.

Identification of a speaker *X* from *N* registered codebooks requires computations of CCS for *N*×*T* number of times as given by Equation (2).

$$Speaker\ \ id = s \quad \arg\min_{1 \leq s \leq N} (D(X, C_s)) \qquad (2)$$

Using total distortion in Equation (2) instead of average distortion used in [4], the best matching registered speaker is decided through Algorithm 3 that computes CCS for less number of times than $N \times T$.

Algorithm 3: Computing minimum speaker distortion using VSP

1: Set $D' = \infty; s = 1; vi = 1;$

2: Do

2.1: Set $i = 1; D = 0$

2.2 Do

2.2.1: Set $D = D + e(x_i, C_s)$

2.2.2 $if\ D \geq D'\ \ goto\ 2.4:$

2.2.3 Set $i = i + 1$

while $i \leq T$

▼ Update currently best distortion and speaker Id

2.3 Set $D' = D; si = s$

2.4 Set $s = s + 1;$

while $s \leq N$

3: Output Test Speaker id = *si*;

Algorithm 3 improves decision about best candidate speaker by updating speaker index *si* and the minimum total distortion D' which are initialized as 1 and ∞, respectively. The pair is updated each time $D(X, C_s) \leq D'$ condition becomes true. Algorithm 3 executes a full CCS or an algorithm of PDE variant to compute $e(x_i, C_s)$. Algorithm 3 avoids redundant evaluation of $e(x_i, C_s)$ by terminating computation of $D(X, C_s)$ for the current C_s as $D \geq D'$ condition becomes true at $i < T$. Hence CCS is actually performed for $N \times T'$ number of times where T' is average of i values when $D \geq D'$ condition occurs for *N* codebooks. Effectively it results in vector sequence pruning (VSP) for unlikely codebooks since $T' < T$.

4. Experiment

4.1 Performance Parameters and Evaluation

In this paper NIST-1999 [10] and TIMIT [11] speech data were used for the speedup experimentations. MFCC feature vector of dimension *d* = 15 is used. Codebook sizes studied were 32, 64, 128, 256, 512, 1024 and 2048. Further details of data selection and feature extraction are included in the appendix. All programs for feature extraction, LBG algorithm and distortion computation were made using Microsoft Visual C# 2008. Hardware used for performance testing was a HP Compac DX7400 Microtower with Intel(R)

Core(TM)2 Duo CPU E6550 @2.33 GHz with 2 Giga byte RAM installed. Operating system used was Windows Vista Business version (2007). Speaker identification time was computed using 'DateTime.Now' function of Microsoft.NET framework. Already extracted feature vectors and trained codebook stored in the hard were used for this purpose.

4.2 Results and Discussion

Accuracy test results of different experiments conducted on TIMIT and NIST-1999 speech data for close-set speaker identification for full CCS, PDE, CSPDE, CPDE and TCPDE are shown in Table 2. Accuracy results are based on testing all speakers in the two databases. Although, the accuracy increases with increase in codebook size, over fitting degradation effect is seen for codebook size 1024 for TIMIT data set. Experiments conducted for NIST-1999 in [4] used 120 seconds long speech samples for training and 30 seconds long ones for testing. All 692 test samples used in [4] belong to 205 target speakers only, while training and testing samples used in this paper are 60 seconds long. This explains comparatively different accuracy and absolute identification times for NIST-1999 listed in this paper since hardware used is also different. Using training and testing sample selection of NIST - 1999 database same as ours for only 30 registered speakers [12] reported an accuracy of 77% with GMM-UBM and graph matching techniques. Comparably we achieved 74.35% accuracy for 230 NIST-1999 registered speakers. Accuracies are better than [7] for TIMIT data primarily due to larger size of feature vectors used in this paper.

Table 2: Accuracy of VQ systems

Codebook Size	Accuracy %	
	TIMIT	NIST'99
32	87.14	65.65
64	97.30	70.43
128	98.89	71.74
256	99.84	73.91
512	100.00	73.91
1024	99.84	73.48
2048	--	74.35

In order to compare approximation performance of PDE, CSPDE, CPDE and TCPDE, we counted the frequency percentage of hitting the closest vectors during CCS for 32 indexes of the code vectors for 230 codebooks of NIST-1999 data. Average behavior of hitting closest code vector and the scan order of code vectors is depicted in Figure 4.

It shows that 55% of times, closest code vectors were correctly identified while scanning the first code vector through CPDE and TCPDE. This is major improvement as compared to PDE and CSPDE for which the frequency of hitting the correct closest code vector at the start of CCS is 4.2% and 3.5%, respectively. In Figure 4 the triangular regions show the improvement of TCPDE over CPDE delaying in locating the closest code vector during CCS.

Incremental performance of PDE, CSPDE, CPDE and TCPDE of hitting the closest code vector with the progress in CCS is plotted in Figure 5. It shows that on the average for TCPDE 90% of times decision about closest code vector is finalized before half scan of the codebooks while corresponding values for CPDE, CSPDE and PDE are 74%, 64% and 45%. The higher values relate to better approximation achieved by the algorithm that causes greater elimination.

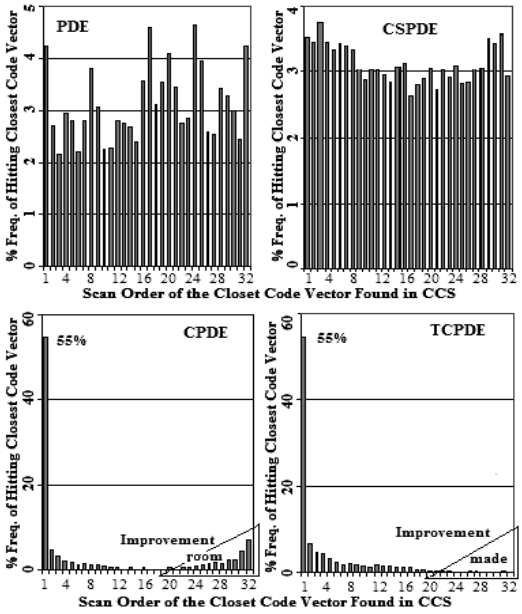

Fig. 4: Average performance of PDE, CSPDE, CPDE and TCPDE on hitting closest code vector earlier in 32 sized codebooks of NIST-1999 data

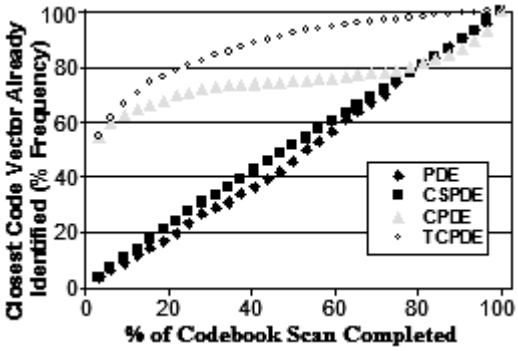

Fig.5 Comparison of PDE, CSPDE, CPDE and TCPDE for correct selection of closest code vector for 32 sized codebooks of NIST'99 data

The complexity order of execution time of VQ systems implies that the basic core-steps are performed $dMNT$ times to identify a test speech speaker. Reduction in the number of times the basic core-steps are performed is the measure of the partial elimination capability of each algorithm proposed in this paper. Table 3 and Table 4 show average number of times the core-steps are avoided as compared to full CCS for registered speakers of TIMIT and NIST-1999 data respectively. The values of $(d-d')/d$ listed in the tables are computed by actually counting the executions of the basic steps for all the test speech samples of the registered speakers for both corpora. Size M, entropy H and normalized entropy $H/\log_2 M$ of each codebooks studied are listed in Table 3 and Table 4 for both corpora. Where $H = -\sum_{m=1}^{M} P_m \log_2 P_m$ and P_m is fraction of training vectors in each cluster of the codebook.

Table 3 and Table 4 show that normalized entropy remains consistent against change in codebook size. This trend is contrary to [9] which indicate that normalized entropy depends only on dimension *d* of MFCC vectors. The tables show that reduction in the number of times core-steps are performed by CSPDE, CPDE and TCPDE with respect to PDE decreases with increase in codebook. This is partly due to the fact that reduction capability of PDE increases with increase in codebook size. The proposed algorithms CPDE and TCPDE cause substantially reduced d' and increase expected speedup factor as compared to CSPDE. Although entropies of code vectors for NIST-1999 data are larger than TIMIT data, CSPDE results in smaller d' for NIST-1999 than for TIMIT.

Table 3: Elimination performance of PDE variants for TIMIT data

Search Algo Used	$\frac{d-d'}{d}$ %	Expected Speedup Factor	$\frac{M}{\frac{H}{H/\log_2 M}}$
PDE	58.67	1	32
CSPDE	69.65	1.36	4.80
CPDE	76.73	1.78	0.96
TCPDE	76.97	1.80	
PDE	65.37	1	64
CSPDE	75.56	1.42	5.77
CPDE	79.97	1.73	0.96
TCPDE	80.28	1.76	
PDE	70.86	1	128
CSPDE	79.39	1.41	6.73
CPDE	82.27	1.64	0.96
TCPDE	82.61	1.68	
PDE	75.02	1	256
CSPDE	82.25	1.41	7.69
CPDE	83.93	1.56	0.96
TCPDE	84.28	1.59	
PDE	77.88	1	512
CSPDE	84.32	1.41	8.61
CPDE	85.15	1.49	0.96
TCPDE	85.49	1.53	
PDE	79.56	1	1024
CSPDE	85.72	1.43	9.46
CPDE	85.97	1.46	0.95
TCPDE	86.31	1.49	

The expected speedup factor values with respect to native PDE listed in Table 3 and Table 4 are calculated while ignoring the cost of extra computation incurred on management of code vector indices and that incurred on conditional branching. Ignoring time for feature vector extraction, Table 5 shows actual speedup

performance based on average speaker identification time for both corpora for the algorithms proposed to guide reimplementation of the techniques in real world systems. In Table 5 speedup factor of CSPDE with respect to plain PDE decreases with codebook size for TIMIT data but unusually high for codebook size 1024. For NIST-1999 data, the speedup factor also decreases with increase in codebook size but it is unusually low for codebook size 64. CPDE and TCPDE outperform CSPDE for both corpora for all codebook sizes studied.

Table 4: Elimination performance of PDE variants for NIST-1999 data

Search Algo Used	$\frac{d-d'}{d}$ %	Expected Speedup Factor	$\frac{M}{H/\log_2 M}$
PDE	53.91	1	32
CSPDE	70.10	1.54	4.90
CPDE	78.08	2.10	0.98
TCPDE	78.30	2.21	
PDE	61.00	1	64
CSPDE	75.19	1.57	5.88
CPDE	80.83	2.03	0.98
TCPDE	81.08	2.06	
PDE	66.95	1	128
CSPDE	79.00	1.57	6.86
CPDE	82.74	1.92	0.98
TCPDE	83.02	1.95	
PDE	71.69	1	256
CSPDE	81.62	1.54	7.85
CPDE	84.21	1.79	0.98
TCPDE	84.42	1.82	
PDE	75.43	1	512
CSPDE	83.59	1.50	8.82
CPDE	85.27	1.67	0.98
TCPDE	85.58	1.70	
PDE	78.14	1	1024
CSPDE	85.09	1.47	9.77
CPDE	86.13	1.58	0.98
TCPDE	86.42	1.61	
PDE	80.00	1	2048
CSPDE	86.24	1.45	10.66
CPDE	86.71	1.51	0.97
TCPDE	87.07	1.55	

Table 5: Time based average speedup performance of CSPDE, CPDE and TCPDE compared with PDE

Code Book Size	Search Algo Used	TIMIT Data		NIST-1999 Data	
		ID Time (S)	Speedup Factor	ID Time (S)	Speedup Factor
32	PDE	0.60	1	1.48	1
	CSPDE	0.55	1.09	1.24	1.13
	CPDE	0.44	1.37	0.95	1.56
	TCPDE	0.40	1.48	0.85	1.74
64	PDE	1.03	1	2.52	1
	CSPDE	0.96	1.08	2.21	1.05
	CPDE	0.76	1.37	1.65	1.53
	TCPDE	0.71	1.45	1.57	1.61
128	PDE	1.81	1	4.43	1
	CSPDE	1.74	1.04	4.01	1.11
	CPDE	1.35	1.34	2.93	1.51
	TCPDE	1.26	1.43	2.76	1.61
256	PDE	3.24	1	7.81	1
	CSPDE	3.15	1.03	7.39	1.06
	CPDE	2.50	1.29	5.39	1.45
	TCPDE	2.33	1.39	5.06	1.54
512	PDE	6.00	1	14.22	1
	CSPDE	5.80	1.03	13.76	1.03
	CPDE	4.84	1.24	10.37	1.37
	TCPDE	4.48	1.34	9.71	1.46
1024	PDE	11.07	1	25.77	1
	CSPDE	10.48	1.06	25.09	1.03
	CPDE	9.13	1.21	19.83	1.3
	TCPDE	8.45	1.31	18.44	1.4
2048	PDE	--	--	47.19	1
	CSPDE			45.57	1.04
	CPDE			38.24	1.23
	TCPDE			35.23	1.34

CPDE is up to 37% and 56% faster than PDE for TIMIT and NIST-1999 data respectively. Figure 4 shows that CPDE hits the closest code vector after half scan of the codebook for 23% of time. This drawback is reduced by TCPDE through toggling between clockwise and anti-clockwise directions. That is why TCPDE is up to 48% and 74% faster than PDE for the respective corpora. Better speed of CPDE and TCPDE than that of CSPDE empirically proves the existence of

M favorable scan orders that are temporally selectable to maximize the utilization of stationarity in speech signal. Speedup results shown in Table 3 to Table 5 are without vector sequence pruning. In order to compare our framework for ASI speedup with [4] we combine VSP individually with CPDE and TCPDE, respectively represented as VSPCPDE and VSPTCPDE. Table 6 shows speedup performance of VSPCPDE and VSPTCPDE for TIMIT and NIST-1999 databases with respect to baseline full search.

Table 6: Average performance of VSPCPDE and VSPTCPDE

Code Book Size	Search Algorithm Type	TIMIT Data		NIST-1999 Data	
		Time (S)	Speedup Factor	Time (S)	Speedup Factor
32	Baseline	1.40	1	3.19	1
	VSPCPDE	0.40	3.53	0.80	3.97
	VSPTCPDE	0.37	3.77	0.76	4.19
64	Baseline	2.70	1	6.22	1
	VSPCPDE	0.67	4.01	1.38	4.52
	VSPTCPDE	0.62	4.37	1.3	4.78
128	Baseline	5.25	1	12.18	1
	VSPCPDE	1.19	4.41	2.44	4.98
	VSPTCPDE	1.10	4.77	2.29	5.32
256	Baseline	10.52	1	24.16	1
	VSPCPDE	2.18	4.83	4.47	5.41
	VSPTCPDE	2.01	5.23	3.9	6.20
512	Baseline	21.08	1	48.37	1
	VSPCPDE	4.17	5.05	8.58	5.64
	VSPTCPDE	3.83	5.50	8.01	6.04
1024	Baseline	42.19	1	95.93	1
	VSPCPDE	7.87	5.36	16.41	5.85
	VSPTCPDE	7.28	5.79	15.3	6.27
2048	Baseline			191.18	1
	VSPCPDE	--	--	31.50	6.07
	VSPTCPDE			29.11	6.57

The speedup factors shown in Table 6 for NIST-1999 data are better than those for TIMIT data for corresponding codebook size. The combinations VSPCPDE and VSPTCPDE have same accuracies as given in Table 2 for the respective corpora and the codebook size. The speedup factors of VSPCPDE and VSPTCPDE in Table 6 are double than those for VPT combined with speaker pruning as reported [4].

5. Conclusions

A framework for speeding up VQ based real-time speaker identification without accuracy loss has been presented. The innate stationarity in test feature vector sequence has been capitalized to substantially improve partial elimination as compared to native PDE and CSPDE. For this, implicit approximation scheme of selecting the closest code vector from previous CCS as first candidate is proposed. Departing from Voronoi view of VQ codebook, proximity insight of LBG arrangement of code vector indexes has been utilized to propose CPDE for a higher level of elimination through circular scan orders of code vectors. CPDE is faster than simple PDE up to 37% for TIMIT and up to 56% for NIST-1999 data, respectively. The delayed hitting of the closest code vectors existing on the other side of circular direction is avoided by proposed TCPDE through toggling between clockwise and anti-clockwise directions. TCPDE is faster than typical PDE up to 48% for TIMIT and up to 74% for NIST-1999 data, respectively. Better speed of CPDE and TCPDE than that of CSPDE empirically indicates the existence of M favorable scan orders that are temporally selectable to maximize utilization of stationarity in speech signal. Vector sequence pruning has also been utilized for codebooks proving unlikely in VSPCPDE and VSPTCPDE. For TIMIT and NIST-1999 data speedup factors achieved by VSPCPDE on the average are up to 5.36 and 6.07 respectively as compared to baseline full search. The speeding up factors of VSPTCPDE for both the data are up to 5.8 and 6.6 respectively.

6. Acknowledgements

Funding support of University of Engineering and Technology, Lahore, Pakistan, is highly appreciated. Public service of Linguistic Data Consortium, University of Pennsylvania, USA, for providing speech data is gratefully acknowledged.

7. References

[1] T. Quatieri, Discrete-time Speech Signal Processing Principles and Practice, Pearson Education, 2002

[2] T. Kinnunen, H. Li, An Overview of Text-Independent Speaker Recognition: from Features to Supervectors, Speech Communication, Elsevier, 52,(1), (2010) pp.12-40

[3] A. Glaser, and F. Bimbot, Steps Towards the Integration of Speaker Recognition in Real-world Telecom Applications, Proc., Int. Conference on Spoken Language Processing, (ICSLP) Sydney, NSW, Australia,1998.

[4] T. Kinnunen, E. Karpove, and P. Franti, Real-Time Speaker Identification and Verification, IEEE Transactions on Audio and Language Processing, January 14, (1), (2006) pp. 277-288.

[5] H. Bei, R. Gray, An Improvement of the Minimum Distortion Encoding Algorithm for Vector Quantization, IEEE Transactions on Communication, 33,(10), (1985) pp.1132-1133

[6] V. Ramasubramanian and K. Paliwal, Fast Nearest-Neighbor Search Algorithms Based on Approximation-Elimination Search, Pattern Recognition, 33,(9), (2000) pp.1497--1510

[7] M. Afzal, and S. Haq, Accelerating Vector Quantization Based Speaker Identification, Journal of American Science, 6, (11), (2010) pp.1046-1050

[8] D. Cheng, A. Gersho, B, Ramamurthi and Y. Shoham, Fast Search Algorithms for Vector Quantization and Pattern Matching, Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 1, (1984) pp. 9.11.1-9.11.4

[9] K. Paliwal and V. Ramasubramanian, Effect of Ordering the Codebook on the Efficiency of the Partial Distance Search Algorithm for Vector Quantization, IEEE Transactions on Communications, 37,(5), (1989) pp. 538-540

[10] A. Martin and M. Przybocki, The NIST 1999 Speaker Recognition Evaluation-- An Overview, Digital Signal Process., ,10, (2000) pp.1-18. http://www.ldc.upenn.edu/

[11] J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren and V Zue, TIMIT Acoustic-Phonetic Continuous Speech Corpus,. 1993. http://www.ldc.upenn.edu/

[12] V. Hautamäki, T. Kinnunen and P. Fränti, Text-Independent Speaker Recognition Using Graph Matching, Pattern Recognition Letters, 29,(9), (2008) p.1427-1432

[13] D. Salomon, Data compression: the complete reference, Volume 10, 4th Ed, Springer, 2007.

Appendix

A.1 Speech Data Selection

NIST 1999 speaker recognition evaluation corpus [10] was used in the experiments to investigate speedup of proposed framework for ASI. In order to tune parameters of speaker identification system TIMIT [11] speech data, consisting of 630 speakers, was used after down sampling it to 8 kHz using anti-aliasing filter. TIMIT data consists of clean microphone speech. There are 10 speech sample files for each of the speaker, in which 2, 5, 3 files are categorized as 'sa', 'sx' and 'si', respectively. All 'sa' and 'sx' files containing same text read by each speaker were concatenated to extract MFCC feature vectors to generate codebooks. The average duration of the concatenated training samples was 22.4 second. Three 'si' files containing different text read by each speaker were concatenated to extract MFCC test feature vectors. The average duration of test samples was 8.4 second.

NIST-1999 data for one-speaker detection test consist of 230 male speakers whose telephonic conversations were recoded in two different sessions. We converted μ-Law companded speech data into linear PCM format and used 'a' files for training and 'b' files for testing. Average duration of speech per speaker used for training or testing was approximately 60 seconds. The speech sample selection both for TIMIT and NIST-1999 corpora conforms to text independent speaker identification setup. The data selection thus allowed speaker identification testing for all registered speaker for both corpora.

A.2 Feature Extraction and Codebook Generation

The digital speech samples were divided into frames, each of 30 milliseconds duration with 40% shift among consecutive frames. To remove silence, 15% average frame energy threshold was used. Frame energy thresholding reduced the training feature vectors by 8.91% and 8.57% for TIMIT and NIST-1999 data, respectively, while testing vectors were reduced by 5.9% and 5.5%, respectively. Hamming window was applied to each non-silence frame before taking Fast Fourier Transform (FFT) to find magnitude spectrum. Filter bank of 27 triangular filters spectrum approximating to Mel-frequency scale was applied to each frame as given in Equation (3).

$$f_{Mel} = 2595 \ \log_{10}(1 + f_{Lin} / 700) \qquad (3)$$

Where f_{Lin} frequency is on linear scale and f_{Mel} is corresponding frequency in Mel scale. For TIMIT data, output of all 27 triangular filters was processed. Subsequently outputs of triangular filter banks were log compressed to take DCT. Ignoring the first value, next 15 values of DCT cepstrum were selected as 15-dimensional MFCC feature vectors. For telephone conversation based NIST-1999 speech data 15 dimensional MFCC vectors were generated while ignoring first 3 and last 3 triangular filters. Triangular filters bank sized 25, 27, 29, 31 were tried. Filterbank of 31 filters that gave the highest accuracy was selected. MFCC vectors of test samples were computed once and stored for tests. VQ codebooks of sizes $\{2^n \mid 5 \le n \le 10\}$ and $\{2^n \mid 5 \le n \le 11\}$ were computed through LBG algorithm from TIMIT and NIST-1999 data respectively. However, code vectors were sorted in decreasing order of cluster size for experimentation of CSPDE.

List of Acronyms and Symbols

ASI = Automated speaker identification

C = Codebook – set of M code vectors

C_s = Codebook of speaker with index s

CCS = Closest code vector search

CPDE = Circular partial distortion elimination

CSPDE = Cluster size based partial distortion elimination

c = Code vector of a codebook

d = Size of code vector – number of elements in a code vector

D = Total distortion of all test vectors in X with a codebook

D' = Threshold minimum total distortion

DCT = Discrete cosine transform

EUD = Euclidean distance

FFT = Fast Fourier transform

f_{Lin} = Linear frequency

f_{Mel} = Mel frequency

GMM = Gaussian mixture model

H = Entropy of code vectors in a codebook

$H/\log_2 M$ = Normalized entropy of code vectors in a code book.

LBG = Linde Buzo Gray

M = Codebook size--number of c in a C

MFCC = Mel frequency cepstral coefficient

N = Number of registered speakers in an ASI system

NIST = National institute of standard and testing

PDE = Partial distortion elimination

PDEs = Refers to any of PDE, CSPDE, CPDE or TCPDE algorithm

$\mathbb{R}$ = Real number space

σ' = Threshold distance between x and c

TCPDE = Toggling circular partial distortion elimination

TIMIT = TI (Texas Instrument) and MIT (Massachusetts Institute of Technology)

VPT = Vantage point tree

VQ = Vector quantization

VSP = Vector sequence pruning

vi = Code vector index best matched with previous test vector

T = Number of x in X

$\tilde{T}$ = Number of $\tilde{x}$ in $\tilde{X}$

X = Vector sequence for speaker testing

$\tilde{X}$ = Vector sequence for speaker training

x_i = An i-th test vector of X

$\tilde{x}_{\tilde{i}}$ = An $\tilde{i}$-th training vector of $\tilde{X}$