2Kbps κωδικοποίηση φωνής για κατανεμημένη αναγνώριση φωνής από PDA σε πραγματικό χρόνο

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
Τμήμα Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών
Εργαστήριο Τηλεπικοινωνιών Πληροφορίας και Δικτύων

Περακάκης Μανόλης
Χανιά , Μάρτιος 2003

Αφιερώνεται στους γονείς μου
Τηλέμαχο και Γαλήνη Περακάκη
για την αγάπη και την
υποστήριξή τους

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον υπεύθυνο της διπλωματικής εργασίας Διγαλάκη Βασίλειο για την συμμετοχή μου στο πολύ ενδιαφέρον αυτό θέμα, καθώς και για την υποστήριξή του στην περάτωση της εργασίας.

Αυτό το κείμενο

Αυτό το κείμενο γράφτηκε σε LaTeX 2_ε με τη βοήθεια ελληνικών γραμματοσειρών, σε περιβάλλον Linux. Οι εικόνες του πρώτου κεφαλαίου είναι από την εργασία [1], ενώ οι υπόλοιπες είναι φτιαγμένες σε xfig. Όλοι οι λατινικοί όροι είναι στοιχειοθετοιμένοι με italics. Για όποια σχόλια ή ερωτήσεις, παρακαλώ επικοινωνείστε μαζί μου στο : `perak@telecom.tuc.gr`

Ανάπτυξη κώδικα

Η ανάπτυξη των τεχνικών που περιγράφονται σε αυτή την εργασία έγιναν στα συστήματα Decipher & Yarrow. Και τα δυο είναι copyright (©) του S.R.I. (Standford Research Institute).

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Εισαγωγή

Τα τελευταία χρόνια είμαστε όλοι μάρτυρες μιας έκρηξης στο χώρο των δικτύων που οδήγησε στην ραγδαία εξάπλωση του *World Wide Web*, κάνοντας έτσι δυνατή την πρόσβαση σε ένα τεράστιο όγκο πληροφοριών. Ανάλογες ραγδαίες εξελίξεις βλέπουμε να συντελούνται και στο χώρο των ασύρματων επικοινωνιών με την εμφάνιση συσκευών όπως *pdas* (*Personal Digital Assistants*) και κινητά νέας γενεάς με δυνατότητα πρόσβασης σε ασύρματα δίκτυα δεδομένων (*802.11, GPRS*), καθιστώντας έτσι δυνατή την πρόσβαση σε πληροφορίες από οπουδήποτε, οποτεδήποτε.

Η φωνή αποτελεί τον πιο φυσικό τρόπο επικοινωνίας και επομένως μπορεί να αποτελέσει ένα πολύ φιλικό τρόπο χρήσης των παραπάνω συσκευών. Ήδη στις μέρες μας, η πιο σημαντική εφαρμογή αναγνώρισης φωνής στο χώρο των τηλεπικοινωνιών, είναι η πρόσβαση σε πληροφορίες μέσω δικτύου φωνής (*PSTN* ή *GSM*) από σταθερά ή κινητά τηλέφωνα. Είναι επόμενο λοιπόν, ότι η δυνατότητα πρόσβασης σε συστήματα αναγνώρισης φωνής μέσω δικτύων δεδομένων (επιπλέον των δικτύων φωνής), θα κάνει τις ήδη υπάρχουσες υπηρεσίες διαθέσιμες στον ολοένα αυξανόμενο αριθμό συσκευών με πρόσβαση σε δίκτυα δεδομένων.

Για να γίνει κάτι τέτοιο εφικτό, προτείνουμε το μοντέλο πελάτη-εξυπηρετητή όπου ο πελάτης είναι οποιαδήποτε συσκευή ικανή να παράγει και να αποστείλει τις παραμέτρους της φωνής στον εξυπηρετητή μέσω δικτύου δεδομένων, ο οποίος αναλαμβάνει το 'δύσκολο' έργο της αναγνώρισης. Το μοντέλο αυτό ονομάζεται κατανεμημένη αναγνώριση φωνής (*distributed speech recognition*).

Η εργασία αυτή επικεντρώνεται στην εύρεση βέλτιστων τρόπων μείωσης της απαιτούμενης πληροφορίας που απαιτείται για την αναπαράσταση των παραμέτρων της φωνής, με τη δημιουργία ενός σχήματος κωδικοποίησης με στόχο την καλύτερη δυνατή συμπίεση φωνής προς αναγνώριση. Μειώνοντας τον απαιτούμενο ρυθμό μετάδοσης δεδομένων σε μόλις *2kbps* καθιστούμε δυνατή την εφαρμογή της αναγνώρισης φωνής σε δίκτυα δεδομένων με ελάχιστο κόστος και περιγράφουμε μια τέτοια υλοποίηση, όπου ο πελάτης είναι μια *Java*

εφαρμογή ικανή να κωδικοποιεί τη φωνή σε πραγματικό χρόνο σε συσκευές όπως *pdas*.

Η τεκμηρίωση της εργασίας χωρίζεται σε 6 κεφάλαια :

Κεφάλαιο 1 Γίνεται μια σύντομη εισαγωγή στα συστήματα αναγνώρισης φωνής. Δίνεται η συνολική εικόνα ενός τέτοιου συστήματος, ενώ στη συνέχεια μελετώνται τα σημαντικότερα υποσυστήματα, όπως ο μηχανισμός δημιουργίας παραμέτρων φωνής, τα ακουστικά μοντέλα καθώς και η κατάλληλη επιλογή αυτών για την περίπτωση αυτής της εργασίας.

Κεφάλαιο 2 Στο πρώτο μέρος αυτού του κεφαλαίου, δίνεται το θεωρητικό υπόβαθρο που απαιτείται για την κατανόηση βασικών εννοιών, όπως οι πηγές και τα μοντέλα πληροφορίας και οι θεωρίες ρυθμού-απώλειας και κωδικοποίησης πηγής χωρίς απώλειες. Παρουσιάζονται επίσης οι τεχνικές κβαντισμού με έμφαση σε τεχνικές διανυσματικού κβαντισμού, ενώ ακολουθεί μια καταγραφή των τεχνικών κωδικοποίησης φωνής.

Στο δεύτερο μέρος δίνεται μια αναλυτική περιγραφή του σχήματος κωδικοποίησης φωνής αυτής της εργασίας, η οποία αποσκοπεί στην συμπίεση φωνής προς αναγνώριση. Αναλύονται οι αλγόριθμοι και περιγράφονται οι διαδικασίες εκπαίδευσης, κβαντισμού και πειραμάτων. Τέλος παρατίθενται τρόποι επιλογής των παραμέτρων του σχήματος κωδικοποίησης και η αξιολόγηση του. Το σχήμα κωδικοποίησης βασίζεται στην ιδέα της διανυσματικής κβαντοποίησης των διανυσμάτων των παραμέτρων της φωνής.

Κεφάλαιο 3 Παρουσιάζεται η υλοποίηση της κωδικοποίησης με το σύστημα *Yarrow*. Επεκτείνοντας το αρχικό σύστημα *Yarrow* με το σχήμα κωδικοποίησης που αναπτύξαμε, το καθιστούμε μια ολοκληρωμένη πλατφόρμα ανάπτυξης εφαρμογών κωδικοποίησης φωνής. Σε επίπεδο κώδικα περιγράφονται αναλυτικά οι βιβλιοθήκες που αναπτύχθηκαν καθώς και οι αλλαγές που έγιναν στο σύστημα αναγνώρισης *Decipher* για συνεργασία με το *Yarrow*. Σε επίπεδο εφαρμογής αναλύεται η χρήση του *Yarrow* για τα πειράματα της αξιολόγησης του σχήματος κωδικοποίησης τα οποία περιγράφονται στο Κεφάλαιο 4.

Κεφάλαιο 4 Παρουσιάζονται τα πειράματα κωδικοποίησης που έγιναν με τα συστήματα φωνής *Yarrow* και *Decipher*. Περιγράφονται πειράματα βαθμωτού και διανυσματικού κβαντισμού, ο αλγόριθμος *bit allocation* για την εύρεση του επιτρεπόμενου ορίου συμπίεσης χωρίς απώλειες στην

απόδοση αναγνώρισης και η επίδραση του θορύβου στην κωδικοποίηση. Περισσότερα πειράματα υπάρχουν στο Παράρτημα Α στο τέλος της εργασίας.

Κεφάλαιο 5 Παρουσιάζονται οι εφαρμογές αναγνώρισης φωνής σήμερα, ενώ στη συνέχεια γίνεται μια εκτενή ανάλυση για τις εξελίξεις σε θέματα δικτύων δεδομένων (κυρίως ασύρματων, όπως κινητής τηλεφωνίας 2.5 και 3ης γενεάς). Αυτό γίνεται, γιατί τα ασύρματα δίκτυα ιδιαίτερα, αναμένεται να αποτελέσουν ιδανικό περιβάλλον για ανάπτυξη εφαρμογών που θα εκμεταλλεύονται την αναγνώριση φωνής με βάση το μοντέλο που προτείνουμε, κυρίως για εφαρμογές ανάκτησης πληροφοριών από ανάλογες συσκευές ασύρματης ανάκτησης πληροφοριών.

Στη συνέχεια περιγράφεται ένα παράδειγμα εφαρμογής του μοντέλου πελάτη - εξυπηρετητή για αναγνώριση φωνής σε δίκτυο δεδομένων. Η εφαρμογή χρησιμοποιεί το σύστημα *Yarrow* στον πελάτη για την κωδικοποίηση της φωνής και το σύστημα αναγνώρισης φωνής *Decipher* στον εξυπηρετητή. Η μετάδοση της κωδικοποιημένης φωνής γίνεται με ρυθμό *2kbps*.

Κεφάλαιο 6 Στο κεφάλαιο αυτό εξετάζουμε το σενάριο πελάτη-εξυπηρετητή του προηγούμενου κεφαλαίου, όπου για πελάτη χρησιμοποιούμε ένα *pda* και η κωδικοποίηση γίνεται σε *live-mode*, όπως θα απαιτούσε οποιαδήποτε πραγματική εφαρμογή. Επειδή συσκευές όπως τα *pdas* δεν διαθέτουν μονάδα κινητής υποδιαστολής, η κωδικοποίηση της φωνής με το *Yarrow* σύστημα του Κεφαλαίου 3 δεν γίνεται σε πραγματικό χρόνο. Γι' αυτό το λόγο, δημιουργήθηκε μια *fixed-point* έκδοση του *Yarrow*, η οποία εξασφαλίζει ίδια επίδοση αναγνώρισης με αυτή του αρχικού συστήματος και είναι πιο γρήγορη. Επιπρόσθετα επεκτάθηκε το *Yarrow*, ώστε να υποστηρίζει κωδικοποίηση σε *live-mode*.

Έτσι η *fixed-point live* έκδοση του *Yarrow* μπορεί να χρησιμοποιηθεί σε *live* αναγνώριση, με τη κωδικοποίηση να γίνεται σε πραγματικό χρόνο μετά το πρώτο δευτερόλεπτο, κάτι που είναι απολύτως αποδεκτό, αφού στη πράξη καμιά πρόταση που θα θέλαμε να αναγνωρίσουμε δεν θα έχει διάρκεια μικρότερη από ένα δευτερόλεπτο.

Κεφάλαιο 1

Σύστημα αναγνώρισης φωνής

1.1 Εισαγωγή

Τα συστήματα αναγνώρισης ομιλίας είναι πολύπλοκα συστήματα που εμπεριέχουν ένα μεγάλο σύνολο γνωστικών πεδίων όπως : στατιστική επεξεργασία σήματος, κατανόηση φυσικής γλώσσας, νευρωνικά συστήματα, αναγνώριση προτύπων, φωνολογία κ.α. Σε αυτό το εισαγωγικό κεφάλαιο πάνω στη αναγνώριση φωνής, θα μελετήσουμε μόνο τα κομμάτια που έχουν να κάνουν περισσότερο με το θέμα αυτής της εργασίας, που είναι η επεξεργασία του σήματος της φωνής (*front-end processing*) και η αναγνώριση με τη χρήση κρυφών μαρκοβιανών μοντέλων (*Hidden Markov Models - HMMs*). Πρώτα όμως, θα δούμε τη συνολική εικόνα ενός συστήματος αναγνώρισης ομιλίας.

1.2 Συνολική εικόνα ενός συστήματος αναγνώρισης ομιλίας

Τα σύγχρονα συστήματα στηρίζονται στις αρχές της στατιστικής αναγνώρισης προτύπων. Οι βασικές μεθοδολογίες της εφαρμογής των παραπάνω αρχών στο πρόβλημα της αναγνώρισης φωνής μελετήθηκαν και εφαρμόστηκαν στη δεκαετία του 70 στην *IBM*. Η συνολική εικόνα έχει ως εξής :

Μια άγνωστη κυματομορφή φωνής μετατρέπεται από τον *front-end* επεξεργαστή σήματος, σε μια ακολουθία από ακουστικά διανύσματα : $Y = y_1, y_2, \ldots, y_t$. Κάθε ένα από τα παραπάνω διανύσματα, αναπαριστά το βραχέως χρόνου φάσμα (*short time speech spectrum*) μιας περιόδου τυπικής διάρκειας *10 msecs*. Έτσι, για παράδειγμα, μια φράση 3 δευτερολέπτων, μπορεί να αναπαρασταθεί από μια ακολουθία 300 τέτοιων διανυσμάτων. Η φράση αποτελείται από μια ακολουθία λέξεων, $W = w_1, w_2, \ldots, w_n$ και είναι ευθύνη

του συστήματος αναγνώρισης να βρει την πιο πιθανή ακολουθία λέξεων W, δεδομένης της ακολουθίας του ακουστικού σήματος Y. Για να γίνει αυτό, χρησιμοποιείται ο κανόνας του *Bayes* :

$$\widehat{W} = \arg\max_{w} P(W|Y) = \arg\max_{w} \frac{P(W)P(Y|W)}{P(Y)} \tag{1.1}$$

Η παραπάνω εξίσωση δηλώνει ότι η πιο πιθανή ακολουθία λέξεων W, είναι εκείνη που μεγιστοποιεί το γινόμενο των *P(W)* και $P(Y|W)$. Ο πρώτος όρος αντιπροσωπεύει την *a-priory* πιθανότητα να παρατηρηθεί το W ανεξάρτητα από το ακουστικό σήμα και καθορίζεται από το γλωσσικό μοντέλο (*language model*). Ο δεύτερος όρος αντιπροσωπεύει την πιθανότητα να παρατηρηθεί η ακολουθία Y, δεδομένης της ακολουθίας W και αυτή η πιθανότητα, καθορίζεται από το ακουστικό μοντέλο (*acoustic model*).

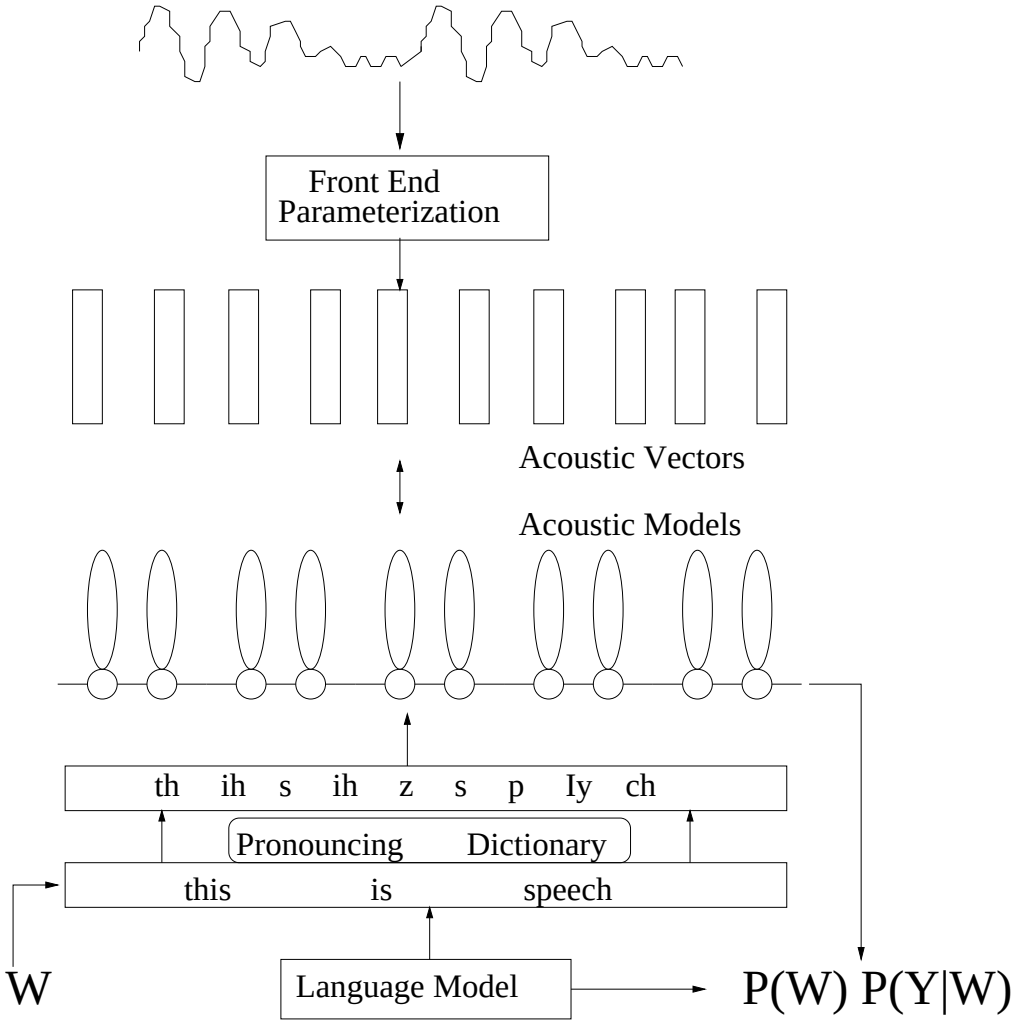
Το Σχήμα 1.1 δείχνει πως τα παραπάνω συνδέονται μεταξύ τους. Η ακολουθία λέξεων « *This is speech* » δίνεται σαν είσοδος και το γλωσσικό μοντέλο υπολογίζει την πιθανότητα *P(W)*. Στη συνέχεια, κάθε λέξη μετατρέπεται σε μια ακολουθία από βασικούς ήχους (*phones*), χρησιμοποιώντας το λεξικό προφοράς (*pronunciation dictionary*). Για κάθε φώνημα υπάρχει ένα αντίστοιχο στατιστικό μοντέλο, το *HMM*. Η ακολουθία των *HMMs* που χρειάζεται για να αναπαραστήσουμε τη φράση συνενώνονται για να σχηματίσουν ένα σύνθετο μοντέλο και υπολογίζεται η πιθανότητα αυτό το μοντέλο να παράγει την ακολουθία Y, δηλαδή η πιθανότητα $P(Y|W)$.

Στη συνέχεια θα δούμε αναλυτικότερα τα εξής :

- Πως είναι σχεδιασμένος ένας *front-end* μηχανισμός για να μπορεί να εξάγει από την κυματομορφή όλη την απαραίτητη ακουστική πληροφορία σε μια μορφή συμβατή με τα *HMM* ακουστικά μοντέλα.
- Πως τα *HMM* μοντέλα αναπαριστούν τη βασική μονάδα αναγνώρισης που είναι το φώνημα.

1.3 Ο front-end μηχανισμός

Μια βασική υπόθεση στην οποία βασίζονται οι αναγνωριστές, είναι ότι το σήμα της φωνής μπορεί να θεωρηθεί σαν στάσιμο (*stationary*-δηλαδή τα χαρακτηριστικά του φάσματος μπορούν να θεωρηθούν σταθερά) σε ένα διάστημα λίγων *msecs*. Το σήμα φωνής χωρίζεται σε ένα σύνολο από διαστήματα (*frames*) και για κάθε διάστημα υπολογίζεται ένα ομαλοποιημένο φάσμα. Τα διαστήματα έχουν συνήθως μήκος *10 msecs* και αλληλεπικαλύπτονται, δίνοντας έτσι ένα



Σχήμα 1.1: **Συνολική εικόνα στατιστικής αναγνώρισης φωνής.** Το διάγραμμα δείχνει τον υπολογισμό της πιθανότητας $P(W|Y)$ της ακολουθίας λέξεων W δεδομένου του ακουστικού σήματος Y. Η *prior* πιθανότητα *P(W)* υπολογίζεται απ' ευθείας από το γλωσσικό μοντέλο (*language model*). Η πιθανότητα $P(Y|W)$ υπολογίζεται χρησιμοποιώντας το σύνθετο *HMM* που αναπαριστά το W και κατασκευάζεται από απλά *HMM* φωνητικά μοντέλα ενωμένα στη σειρά, ανάλογα με τις προφορές των λέξεων που είναι αποθηκευμένες στο λεξικό προφορών.

παράθυρο μεγαλύτερου μήκους. Χρησιμοποιώντας για παράδειγμα παράθυρο *Hamming* και εφαρμόζοντας ανάλυση *Fourier* ή γραμμικής πρόβλεψης (*LPC*) παίρνουμε τις βασικές φασματικές παραμέτρους τις οποίες με διάφορους μετασχηματισμούς μετατρέπουμε σε κάποια κατάλληλη μορφή.

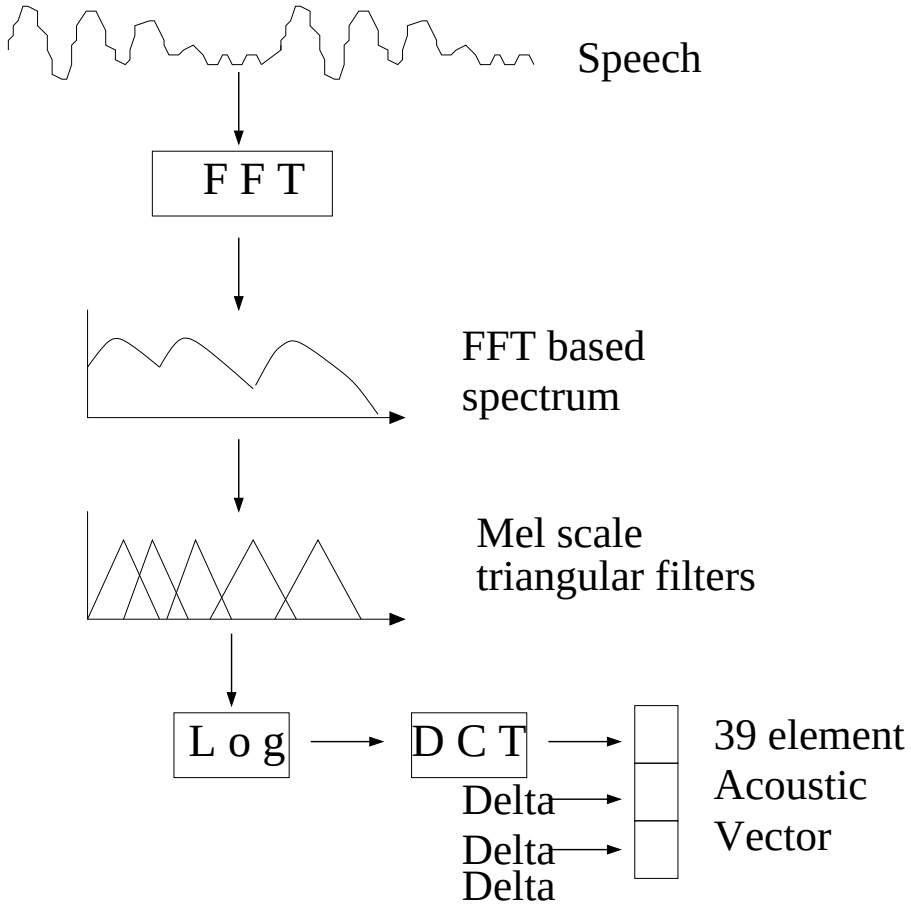
Μια τέτοια μορφή για τα ακουστικά διανύσματα είναι οι *mel-frequency cepstral coefficients* (*MFCCs*). Ένας τρόπος για να υπολογιστούν, φαίνεται στο Σχήμα 1.2. Αφού πάρουμε το φάσμα του σήματος εφαρμόζουμε μια κλίμακα (*mel-frequency*) η οποία σχεδιάζεται ουσιαστικά ώστε να προσεγγίζει την φασματική ανάλυση της ανθρώπινης ακοής, που είναι γραμμική έως τα *1000Hz* και λογαριθμική από κει και πέρα. Η κλίμακα αυτή έχει αποδειχθεί πειραματικά ότι βελτιώνει την ακρίβεια αναγνώρισης. Στη συνέχεια, για να κάνουμε την ισχύ του φάσματος περίπου γκαουσιανή εφαρμόζουμε λογαριθμική συμπίεση. Τέλος, αφού εφαρμόσουμε *Discrete Cosine Transform*, παίρνουμε ένα ακουστικό διάνυσμα που αποτελείται από 12 *cepstral* συντελεστές και την ενέργεια του σήματος. Αν προσθέσουμε τις πρώτες και δεύτερες παραγώγους έχουμε τελικά ένα διάνυσμα μεγέθους 39.

1.3.1 Η διαδικασία κβαντισμού των παραμέτρων του front-end

Το τελικό αποτέλεσμα επομένως, είναι η παραγωγή ενός *MFCC* διανύσματος για κάθε *frame* (δηλαδή για κάθε *10 msecs*) φωνής. Ο βασικός σκοπός αυτής της εργασίας, είναι ο βέλτιστος διανυσματικός κβαντισμός των παραπάνω διανυσμάτων. Η διαδικασία αυτή περιγράφεται εκτενώς σε επόμενο κεφάλαιο. Εδώ πολύ σύντομα θα αναφέρουμε απλά την κεντρική ιδέα που είναι η εξής :

> Το *MFCC* διάνυσμα Y χωρίζεται κατάλληλα σε μικρότερα υποδιανύσματα $y_1, y_2, \ldots, y_L$ και κάθε τέτοιο υποδιάνυσμα κβαντίζεται στο πιο κοντινό σε αυτό υποδιάνυσμα $\widehat{y}_i$, από ένα σύνολο από τέτοια κατάλληλα επιλεγμένα υποδιανύσματα (το σύνολο αυτό ονομάζεται *codebook*).

Το αποτέλεσμα του κβαντισμού είναι η συμπίεση του σήματος (αφού κάθε υποδιάνυσμα αναπαρίσταται από το δείκτη του υποδιανύσματος στο οποίο κβαντίστηκε) και η επιτάχυνση της αναγνώρισης καθώς, όπως θα δούμε και σε επόμενη παράγραφο, μπορεί να χρησιμοποιηθεί μια νέα κατηγορία ακουστικών μοντέλων που εκμεταλλεύεται την ύπαρξη του κβαντισμού των ακουστικών διανυσμάτων στο *front-end*.



Σχήμα 1.2: **Παραγωγή MFCC διανύσματος.** Για να γίνει *pattern matching*, η κυματομορφή μετατρέπεται σε μια ακολουθία ακουστικών διανυσμάτων, τα οποία αντιπροσωπεύουν ένα ομαλοποιημένο λογαριθμικό φάσμα που υπολογίζεται για κάθε *10 msecs* φωνής. Χρησιμοποιώντας μια μη-γραμμική *mel-frequency* κλίμακα και Μετασχηματισμό Διακριτού Συνημιτόνου (*Discrete Cosine Transform - DCT*) βελτιώνουμε την απόδοση. Το πρώτο, έχει σαν αποτέλεσμα την συμπίεση της πληροφορίας στους πρώτους συντελεστές του διανύσματος, ενώ το τελευταίο, έχει σαν αποτέλεσμα την αποσυσχέτιση (*decorrelation*) του σήματος, βελτιώνοντας τις υποθέσεις για στατιστική ανεξαρτησία και επιτρέποντας τη χρήση διαγωνίων πινάκων συνμεταβλητότητας (*Covariance matrices*). Τέλος, για να ενσωματώσουμε δυναμική πληροφορία για το σήμα, προσθέτουμε τις πρώτες και δεύτερες παραγώγους.

1.4 Acoustic Modeling

Τα ακουστικά μοντέλα παρέχουν ένα τρόπο για τον υπολογισμό της πιθανότητας $P(Y|W)$. Μονάδα μοντελοποίησης είναι το φώνημα που αναπαρίσταται από ένα *HMM* (*triphone*[1]). Ένα *HMM* έχει μία κατάσταση εισόδου, μία κατάσταση εξόδου και 3 ενδιάμεσες καταστάσεις. Οι καταστάσεις εισόδου-εξόδου χρησιμεύουν στην ένωση πολλών διαδοχικών *HMMs*, τα οποία σχηματίζουν ένα σύνθετο *HMM*, το οποίο μπορεί να αναπαραστήσει μεγαλύτερες μονάδες, όπως λέξεις ή μια ολόκληρη πρόταση. Το HMM μπορεί να θεωρηθεί σαν μια γεννήτρια ακολουθίας συμβόλων (στη περίπτωσή μας, ακουστικών διανυσμάτων) μοντελοποιώντας παράλληλα μια κρυφή ακολουθία καταστάσεων (αποτελούμενων από τις 3 ενδιάμεσες καταστάσεις). Η πιθανότητα μετάβασης από την κατάσταση i στην κατάσταση j καθορίζεται από την διακριτή πιθανότητα a_{ij}, ενώ η πιθανότητα παραγωγής ενός συμβόλου στη κατάσταση j, καθορίζεται από τη πιθανότητα εξόδου b_j.

Στο Σχήμα 1.3 φαίνεται ένα παράδειγμα της παραπάνω διαδικασίας όπου το μοντέλο μεταβαίνει μέσω της ακολουθίας καταστάσεων $X = 1, 2, 2, 3, 4, 4, 5$, για να παράγει την ακολουθία y_1 έως y_5. Η συνδυασμένη πιθανότητα μιας ακολουθίας διανυσμάτων Y και μιας ακολουθίας καταστάσεων X, δεδομένου κάποιου μοντέλου M (πιθανότητες μετάβασης και εξόδου), υπολογίζεται σαν το γινόμενο των πιθανοτήτων μετάβασης και των πιθανοτήτων εξόδου. Για την ακολουθία καταστάσεων X του σχήματος 1.3 έχουμε ότι :

$$P(Y, X|M) = a_{12}b_1(y_1)a_{22}b_2(y_2)a_{23}b_3(y_3)\dots \tag{1.2}$$

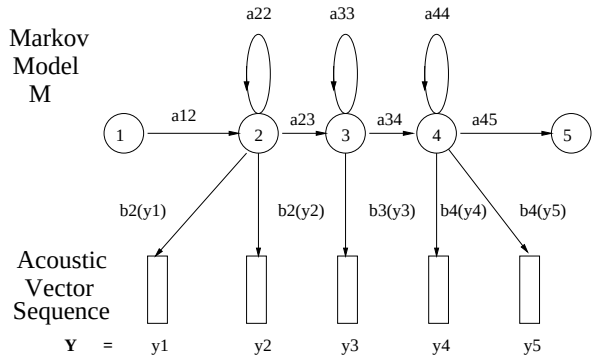
Στην γενική περίπτωση που $X = x(1), x(2), x(3), \dots, x(T)$, η παραπάνω πιθανότητα γίνεται :

$$P(Y, X|M) = a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(y_t)a_{x(t)x(t+1)} \tag{1.3}$$

όπου $x(0)$ είναι η κατάσταση εισόδου του μοντέλου και $x(T+1)$ η κατάσταση εξόδου.

Στην πραγματικότητα, εμείς γνωρίζουμε μόνο την ακολουθία Y, ενώ η ακολουθία X μας είναι άγνωστη, για αυτό και μιλάμε για κρυφές Μαρκοβιανές ακολουθίες. Για τον υπολογισμό λοιπόν της $P(Y|M)$ αρκεί να αθροίσουμε την 1.3 για όλες τις πιθανές μεταβάσεις καταστάσεων. Ένας αποτελεσματικός τρόπος για να γίνει αυτό είναι ο *Forward-Backward* αλγόριθμος.

[1] Κάθε φώνημα μπορεί να υπάρχει σε διαφορετικά *contexts* (συμφραζόμενα) τα οποία καθορίζονται από τα αντίστοιχα *HMMs* (*triphones*). Για τα 45 φωνήματα της αγγλικής γλώσσας υπάρχουν 45^3 *triphones*, από τα οποία μόνο 60.000 χρησιμοποιούνται στη πράξη



Σχήμα 1.3: **HMM μοντέλο.** Ένα HMM μπορεί να θεωρηθεί σαν μια γεννήτρια ακολουθίας συμβόλων (στη περίπτωσή μας, ακουστικών διανυσμάτων) μοντελοποιώντας παράλληλα μια κρυφή ακολουθία καταστάσεων. Μεταβαίνει μεταξύ των καταστάσεων βάσει των πιθανοτήτων μετάβασης και κάθε χρονική περίοδο παράγει ένα καινούριο ακουστικό διάνυσμα σύμφωνα με την κατανομή εξόδου της παρούσας κατάστασης.

1.5 Είδη HMMs

Η επιλογή της κατανομής εξόδου, παίζει σημαντικό ρόλο αφού μοντελοποιεί την μεταβλητότητα του φάσματος της φωνής , σε αντίθεση με το μοντέλο μετάβασης καταστάσεων το οποίο έχει να κάνει με την διάρκεια. Ανάλογα λοιπόν με το πως μοντελοποιούνται οι κατανομές εξόδου, έχουμε τρεις κατηγορίες ακουστικών μοντέλων.

1.5.1 Διακριτά HMMs

Τα πρώτα συστήματα αναγνώρισης φωνής χρησιμοποιούσαν διακριτές κατανομές εξόδου σε συνδυασμό με διανυσματικό κβαντιστή (*Vector Quantizer - VQ*). Έτσι, κάθε ακουστικό διάνυσμα αντικαθίσταται από το *index* (δείκτη) του πιο κοντινού σε αυτού διανύσματος από ένα σύνολο τέτοιων διανυσμάτων (*codebook*), ενώ οι κατανομές εξόδου ισοδυναμούν με *look-up tables* που περιέχουν τις πιθανότητες για κάθε πιθανό δείκτη. Για τη κατανομή εξόδου $b_j(y_t)$ στην κατάσταση j, με y_t το ακουστικό διάνυσμα τη χρονική στιγμή t θα ισχύει, δεδομένου της απεικόνισης του y_t στο διάνυσμα y_m του *codebook* :

$$\sum_{m=1}^{M} b_j(y_m) = 1 \tag{1.4}$$

όπου M είναι το μέγεθος του *codebook*. Το μεγάλο πλεονέκτημα με τα διακριτά μοντέλα είναι ότι υπολογιστικά αυτός ο τρόπος είναι αρκετά αποτελεσματικός από άποψη αναγνώρισης, αφού η πιθανότητα εξόδου δεν χρειάζεται να υπολογιστεί, αλλά ανακτάται μέσω ενός *look-up table*. Αντίθετα το μεγαλύτερο μέρος του χρόνου σπαταλάται στον κβαντισμό, όπου πρέπει να υπολογιστεί η ελάχιστη απόσταση του διανύσματος εισόδου με τα πρότυπα ακουστικά διανύσματα που περιέχονται στο *codebook*. Βλέπουμε λοιπόν ότι ο μηχανισμός *VQ* δρα σαν προ-επεξεργαστής αναγνώρισης (*recognition preprocessor*).

Όμως εκ των πραγμάτων, η ανάλυση του ακουστικού χώρου είναι μικρή (αφού έχουμε πεπερασμένο αριθμό συμβόλων εξόδου) με επιπτώσεις στην ακρίβεια της αναγνώρισης, οι οποίες ενισχύονται και λόγω του κβαντισμού ο οποίος εισαγάγει επιπλέον θόρυβο. Επίσης η χρήση *VQ* με είσοδο 'ολόκληρο' το διάνυσμα ως έχει, απαιτεί πολύ μεγάλα *codebooks* για να καλύψει το ακουστικό χώρο, ακόμα και όταν αυτός παραμένει σχετικά μικρός.

Βέβαια για τις πρώτες εφαρμογές αναγνώρισης φωνής το μοντέλο αυτό λειτουργούσε ικανοποιητικά, αφού εκείνη την εποχή η διαθεσιμότητα σε πόρους όπως μνήμη και υπολογιστική ισχύς ήταν περιορισμένη, ενώ και τα λεξιλόγια ήταν σχετικά μικρά. Καθώς περνούσαν τα χρόνια, με την εισαγωγή όλο και ισχυρότερων υπολογιστικών συστημάτων αλλά κυρίως με τις απαιτήσεις εφαρμογών για πολύ μεγαλύτερο λεξιλόγιο, έγινε φανερό ότι το συγκεκριμένο μοντέλο δεν ήταν πια κατάλληλο.

1.5.2 Συνεχή HMMs - Gaussian Mixture HMMs

Έτσι τα σύγχρονα συστήματα αναγνώρισης χρησιμοποιούν παραμετρικές συνεχείς κατανομές εξόδου που μοντελοποιούν τα ακουστικά διανύσματα απευθείας. Για το σκοπό αυτό συνήθως χρησιμοποιούνται μείγματα γκαουσιανών κατανομών και η πιθανότητα εξόδου (*output propability*) είναι η εξής :

$$b_j(y_t) = \sum_{m=1}^{M} c_{jm} N(y_t; \mu_{jm}, \Sigma_{jm}) \tag{1.5}$$

όπου c_{jm} είναι το βάρος (*weight*) του μείγματος m στην κατάσταση j και $N(y_t; \mu_{jm}, \Sigma_{jm})$ είναι η πολυδιάστατη (*multivariate*) γκαουσιανή με μέση τιμή μ και συνδιακύμανση Σ για το μείγμα m στην κατάσταση j. Συνήθως «παρόμοιες» καταστάσεις μεταξύ *HMMs* χρησιμοποιούν κοινές γκαουσιανές, έτσι ώστε να μειώνεται ο συνολικός αριθμός τους. Παρ' όλα αυτά, ο χρόνος που χρειάζεται τόσο για την εκπαίδευση των παραμέτρων τους, όσο και κατά την διαδικασία της αναγνώρισης, είναι αρκετά μεγαλύτερος σε σχέση με τους αντίστοιχους χρόνους με διακριτά *HMM* μοντέλα.

1.5.3 Μείγματα διακριτών HMMs από κβαντοποίηση συνεχών HMMs

Όπως είναι φανερό από τα παραπάνω, παρουσιάζεται το φαινόμενο τα μειονεκτήματα της μιας κατηγορίας *HMMs* να αποτελούν πλεονεκτήματα για το άλλο είδος και αντίστροφα. Στο εργαστήριο μας υλοποιήθηκε ένα καινούριο είδος ακουστικών μοντέλων που συγκεντρώνει τα πλεονεκτήματα των 2 μεθόδων και προκύπτει με τον εξής συλλογισμό :

Η ιδέα είναι να εφαρμόσουμε το *VQ* όχι απ'ευθείας στα ακουστικά μοντέλα αλλά στο *front-end*. Με την έξοδο του *VQ* να μην είναι πια ο δείκτης, αλλά το ίδιο το πρότυπο διάνυσμα, μπορούμε να πειραματιστούμε μέχρι να καταλήξουμε σε κάποιο σχήμα κωδικοποίησης που να δίνει υψηλή ακρίβεια αναγνώρισης για το ζητούμενο βαθμό συμπίεσης (με χρήση πάντα των *Gaussian Mixture HMMs*). Εφαρμόζοντας το σχήμα κωδικοποίησης στο οποίο καταλήξαμε, κβαντίζουμε σε δείκτη τώρα πια και παράλληλα διακριτοποιούμε τα μείγματα των γκαουσιανών. Τώρα, περιμένουμε το ίδιο επίπεδο ακρίβειας αναγνώρισης (λόγω της ικανοποιητικής ανάλυσης του ακουστικού χώρου που διατηρήσαμε) καθώς και επιτάχυνσης του χρόνου αναγνώρισης (λόγω της αντικατάστασης του υπολογισμού της πιθανότητας εξόδου με ένα απλό *look-up*). Καταλυτικό ρόλο, παίζει το γεγονός ότι το κάθε ακουστικό διάνυσμα 'σπάει' σε κατάλληλα επιλεγμένα μικρότερα διανύσματα (έστω L). Αυτό έχει σαν αποτέλεσμα την ακόμα μεγαλύτερη επιτάχυνση του χρόνου αναγνώρισης, καθώς αντί να χρησιμοποιείται ένα 'τεράστιο' *codebook* κατά τη διάρκεια του κβαντισμού, χρησιμοποιούνται L μικρά *codebooks*.

Βασιζόμενοι λοιπόν στον διανυσματικό κβαντισμό των *MFCC* διανυσμάτων του *front-end* σύμφωνα με τα παραπάνω, η κατανομή εξόδου $b_j(y_t)$, με $y_t = [y_{1t}, y_{2t} \dots y_{Lt}]$ το διάνυσμα που περιέχει τα L υποδιανύσματα είναι :

$$b_j(y_t) = \sum_{m=1}^{M} c_{jm} \prod_{i=1}^{L} P_{mi}(vq(y_{it})) \tag{1.6}$$

όπου c_{jm} είναι το βάρος (*weight*) του μείγματος m στη κατάσταση j και $P_{mi}(vq(y_{it}))$ η πιθανότητα του διακριτού συμβόλου $vq(y_{it})$ για το υποδιάνυσμα i του μείγματος m. Με $vq(y_{it})$ αναπαριστούμε το σύμβολο εξόδου του κβαντισμού του υποδιανύσματος i την χρονική στιγμή t, με άλλα λόγια τον αντίστοιχο δείκτη.

Συγκρίνοντας τις εξισώσεις 1.6 και 1.5, βλέπουμε ότι έχουμε αντικατάσταση της γκαουσιανής με ένα γινόμενο πιθανοτήτων, πράγμα που έχει σαν αποτέλεσμα την αντικατάσταση του υπολογισμού των πιθανοτήτων εξόδου των γκαουσιανών με ένα *look-up* και συνεπώς την επιτάχυνση της αναγνώρισης. Για περισσότερες λεπτομέρειες παραπέμπουμε στην διπλωματική εργασία [18].

1.6 Συμπεράσματα

- Τα συστήματα αναγνώρισης φωνής βασίζονται στην ύπαρξη ενός μηχανισμού παραγωγής 'ακουστικών' διανυσμάτων (*front-end*), ακουστικών μοντέλων που μοντελοποιούν τα φωνήματα χρησιμοποιώντας κρυφά μαρκοβιανά μοντέλα (*HMMs*) και ενός γλωσσικού μοντέλου. Με χρήση στατιστικών και άλλων μεθόδων, γίνεται τελικά η αποκωδικοποίηση (αναγνώριση) της φωνής.

- Η εισαγωγή του κβαντισμού των παραμέτρων του *front-end* έχει σαν βασικό σκοπό την μέγιστη συμπίεση. Αυτό μας επιτρέπει π.χ. την αναγνώριση σε περίπτωση που ο ρυθμός μετάδοσης δεδομένων ενός διαύλου μεταξύ *front-end* και αναγνωριστή είναι περιορισμένος.

- Τα πρώτα συστήματα χρησιμοποιούσαν διακριτά μοντέλα, τελικά όμως επικράτησαν ακουστικά μοντέλα που χρησιμοποιούν γκαουσιανά μείγματα *HMMs*. Για τους σκοπούς αυτής της εργασίας η οποία προτείνει τη χρήση του διανυσματικού κβαντισμού, χρησιμοποιήθηκαν ακουστικά μοντέλα που προκύπτουν με διακριτοποίηση των τελευταίων (*DMHMMs*).

- Πέρα από αυτό όμως, μας δίνεται η δυνατότητα να χρησιμοποιήσουμε το κβαντισμό των παραμέτρων του *front-end* με αναγνωριστή που χρησιμοποιεί τόσο μείγματα συνεχών *HMMs* (στην περίπτωση που ο κβαντισμός υποδιανύσματος δίνει κβαντισμένο υποδιάνυσμα) όσο και μείγματα διακριτών *HMMs* από κβαντοποίηση συνεχών *HMMs* (στην περίπτωση που ο κβαντισμός υποδιανύσματος δίνει τον δείκτη του κβαντισμένου υποδιανύσματος). Στην δεύτερη περίπτωση έχουμε και μεγαλύτερη συμπίεση και ταχύτερη αναγνώριση.

1.7 Παραπομπές

Σε αυτό το κεφάλαιο δόθηκε μια πολύ σύντομη εισαγωγή στο κεφάλαιο που λέγεται αναγνώριση φωνής. Μια πιο πλήρης αλλά και περιεκτική εισαγωγή δίνεται στην εργασία [1], η οποία επίσης περιέχεται και στο βιβλίο [3] για το *HTK* σύστημα. Πιο ειδικά, μια αναλυτική περιγραφή για επεξεργασία φωνής δίνεται στο [2]. Ένα αρκετά αξιόλογο και πλήρες βιβλίο είναι το [4] για το οποίο συνίσταται και το [5]. Στη [4] δίνεται μια ολοκληρωμένη εικόνα του τομέα αναγνώρισης φωνής, από τεχνικές για παραγωγή ακουστικών διανυσμάτων και σύγκρισης προτύπων, μέχρι την εκπαίδευση και προσαρμογή μοντέλων, αλλά και πρακτικές λύσεις για την αποτελεσματικότητα ενός συστήματος αναγνώρισης φωνής.

Κεφάλαιο 2

Κωδικοποίηση φωνής

2.1 Εισαγωγή

Όπως είδαμε στην Εισαγωγή, σκοπός αυτής της εργασίας είναι η εύρεση τρόπου κωδικοποίησης φωνής σε επίπεδα που θα καθιστούν την ανάπτυξη εφαρμογών αναγνώρισης φωνής πιο προσιτή από άποψη κόστους, αφού αυτό είναι συνυφασμένο με το ρυθμό μετάδοσης δεδομένων. Αυτό θα επιτρέψει την χρήση του μοντέλου πελάτη - εξυπηρετητή για εφαρμογές σε περιβάλλον όπως αυτό του διαδικτύου ή των ασύρματων δικτύων, όπου ο πελάτης είναι υπεύθυνος για τη κωδικοποίηση της φωνής.

Στο Κεφάλαιο 1 είδαμε ότι το *front-end* παράγει τις παραμέτρους της φωνής με μορφή ακουστικών διανυσμάτων, τα οποία χρησιμοποιούνται από τα ακουστικά μοντέλα για να βρεθεί η βέλτιστη ακολουθία λέξεων από τον αναγνωριστή. Αναφέρθηκε ότι η χρήση διανυσματικού κβαντισμού επιτυγχάνεται με το κβαντισμό των διανυσμάτων εισόδου του *front-end* στα πρότυπα ακουστικά διανύσματα αναφοράς που περιέχονται στο *codebook*. Επιπώθηκε επίσης, ότι η χρήση του διανυσματικού κβαντισμού επιτρέπει τη χρησιμοποίηση ακουστικών μοντέλων που προκύπτουν από διακριτοποίηση μοντέλων με μείγματα συνεχών κατανομών, τα οποία δημιουργήθηκαν ειδικά για το σκοπό αυτής της εργασίας. Ο συνδυασμός του διανυσματικού κβαντισμού με χρήση των παραπάνω μοντέλων, έχει σαν αποτέλεσμα τη συμπίεση του σήματος της φωνής, αλλά και την επιτάχυνση της αναγνώρισης.

Στο πρώτο μέρος αυτού του κεφαλαίου δίνεται κάποιο θεωρητικό υπόβαθρο, παρουσιάζοντας τεχνικές, όπως την κωδικοποίηση πηγής, τον βαθμωτό και διανυσματικό κβαντισμό και τέλος τις τεχνικές κωδικοποίησης φωνής. Στο δεύτερο μέρος του κεφαλαίου, περιγράφεται αναλυτικά το σχήμα κωδικοποίησης που εφαρμόσαμε το οποίο συνδυάζει τεχνικές κωδικοποίησης φωνής και

κωδικοποίησης πηγής. Αναλύονται οι βασικές έννοιες, οι ορολογίες και οι αλγόριθμοι που χρησιμοποιήθηκαν και περιγράφονται οι διαδικασίες εκπαίδευσης για τη δημιουργία των *codebooks* αλλά και του διανυσματικού κβαντισμού με τη χρήση τους.

> Πρώτα όμως πρέπει να γίνει ξεκάθαρο, ότι σε αντίθεση με τις συνήθεις τεχνικές κωδικοποίησης που σαν σκοπό έχουν την πιστή αναπαραγωγή του σήματος της φωνής, η κωδικοποίηση που εφαρμόζουμε, υπηρετεί ένα διαφορετικό σκοπό : Τη μέγιστη συμπίεση των παραμέτρων της φωνής με σκοπό όχι την πιστή αναπαραγωγή του σήματος, αλλά την καλύτερη δυνατή αναγνώριση.

2.2 Πηγές πληροφορίας και κωδικοποίηση

Τα συστήματα επικοινωνιών σχεδιάζονται για να μεταδίδουν πληροφορία. Σε όλα αυτά τα συστήματα υπάρχει μια πηγή που παράγει την πληροφορία και σκοπός του συστήματος είναι να μεταδώσει την έξοδο της πηγής στον προορισμό της. Όλοι έχουμε εμπειρία από αυτό το μοντέλο : εκπομπές ραδιοφωνικού σήματος, τηλεόρασης, μετάδοση φωνής κ.τ.λ. Προκειμένου να μπορούμε να κάνουμε ανάλυση απόδοσης ενός τέτοιου μοντέλου όμως, χρειάζονται τα κατάλληλα εργαλεία : ποσοτικές μετρικές και κατάλληλα μαθηματικά μοντέλα. Στην ανάπτυξη των τελευταίων βοήθησαν καταλυτικά οι *Hartley, Nyquist & Shannon*.

Η πηγή πληροφορίας παράγει έξοδο που ο προορισμός της δεν την γνωρίζει από πριν - η πληροφορία έχει να κάνει με τη γνώση για κάτι. Η έξοδος της πηγής είναι μια μη-προβλέψιμη, χρονικά μεταβαλλόμενη διαδικασία, άρα μπορεί να μοντελοποιηθεί σαν μια τυχαία(στοχαστική) διαδικασία. Το χαρακτηριστικό των πηγών αυτών, είναι ότι μπορούν να μοντελοποιηθούν σαν *bandlimited* διαδικασίες. Για παράδειγμα, το σήμα της φωνής έχει σχεδόν όλη την ισχύ του στην μπάντα 300-3400Hz. Εκμεταλλευόμενοι αυτό το γεγονός, μπορούμε να κάνουμε δειγματοληψία σε συχνότητα *Nyquist* και πλέον θα έχουμε να κάνουμε με διακριτού χρόνου στοχαστικές διαδικασίες.

2.2.1 Μοντέλο πληροφορίας

Ένα απλό μοντέλο πληροφορίας είναι η πηγή να μοντελοποιείται από μια διακριτού χρόνου στοχαστική διαδικασία $\{X_i\}_{i=-\infty}^{\infty}$. Το αλφάβητο στο οποίο ορίζονται οι τυχαίες μεταβλητές X_i μπορεί να είναι διακριτές ή συνεχείς (π.χ. δειγματοληπτημένη φωνή), ενώ οι στατιστικές ιδιότητες της διαδικασίας καθορίζονται από την ίδια τη πηγή. Για τη φωνή, ένα τέτοιο μοντέλο είναι μια

διακριτού χρόνου και πλάτους τυχαία διαδικασία, όπου όλα τα X_i παράγονται ανεξάρτητα (διαδικασία χωρίς μνήμη-*memoryless process*) και με την ίδια κατανομή.

Ας υποθέσουμε ότι η X παίρνει τιμές από ένα σύνολο $A = \{a_1, a_2, ..., a_M\}$ με πιθανότητες $p_i = p(X = a_i)$, $i = 1, 2...M$. Μια παρατήρηση για την 'ποιοτική' μέτρηση της πληροφορίας, είναι ότι αυτή πρέπει να μειώνεται όσο αυξάνει η πιθανότητα παρατήρησης του συμβόλου εξόδου. Για παράδειγμα, η είδηση ότι μέσα στον Ιούλιο έχουμε βροχή περιέχει περισσότερη πληροφορία από την είδηση ότι έχουμε ηλιοφάνεια. Μια άλλη παρατήρηση, είναι ότι μια μικρή αλλαγή στη πιθανότητα, έχει μικρή επίπτωση στο ποσό της πληροφορίας. Με άλλα λόγια, μια μετρική πληροφορίας θα πρέπει να είναι μια συνεχής και μειούμενη συνάρτηση της πιθανότητας της εξόδου της πηγής.

Μια τέτοια 'συμπεριφορά' παρέχεται από την : $-log(p_i)$ που ονομάζεται *self-information*. Η εντροπία μιας πηγής αποτελεί μια καλή μετρική σύμφωνα με τις παραπάνω παρατηρήσεις και ορίζεται από τη σχέση : $H(X) = -\sum_{i=1}^{N} p_i log p_i$ Η εντροπία της πηγής, μας δίνει ένα αρκετά ακριβές όριο του ρυθμού στον οποίο μπορεί να συμπιεστεί η έξοδος της πηγής, για ικανή ανακατασκευή του σήματος χωρίς απώλειες. Αυτό αποδεικνύεται από το θεώρημα κωδικοποίησης πηγής του *Shannon*, σύμφωνα με το οποίο για να έχουμε ικανή ανακασκευή του σήματος χωρίς απώλειες, θα πρέπει να ισχύει : $R > H$, με R το ρυθμό μετάδοσης (συμπίεση).

Τεχνικές που 'σέβονται' το παραπάνω κριτήριο, δηλαδή κωδικοποίηση χωρίς απώλειες, (ονομάζονται *lossles coding* ή και *entropy coding*) είναι η κωδικοποίηση *Huffman* και *Ziv-Lempel*. Συχνά όμως, αποφασίζουμε ότι θέλουμε να συμπιέσουμε την έξοδο της πηγής περισσότερο, με κόστος κάποια μικρή απώλεια πληροφορίας (παραμόρφωση-*distortion*). Σε αυτή τη περίπτωση, λέμε ότι έχουμε κβαντισμό (*quantization*) ή κωδικοποίηση με απώλειες (*lossy coding*). Στην επόμενη παράγραφο αναλύονται δυο τέτοιες τεχνικές.

2.3 Τεχνικές κβαντισμού

Ένα σύστημα κβαντισμού απεικονίζει μια είσοδο σε μια έξοδο, χρησιμοποιώντας κάποιο κατάλληλο μηχανισμό απεικόνισης. Για παράδειγμα, απεικονίζοντας ένα πραγματικό αριθμό σε μια *float* μεταβλητή, έχουμε ήδη κβαντίσει τον πραγματικό αριθμό σε μια από τις διαθέσιμες τιμές (στάθμες) και έχουμε ήδη εισαγάγει θόρυβο. Απλά, επειδή ο αριθμός διαθέσιμων τέτοιων τιμών είναι αρκετά μεγάλος, ο θόρυβος είναι πολύ μικρός και δεχόμαστε ότι δεν έχουμε πρακτικά απώλεια αναπαράστασης κατά την απεικόνιση αυτή. Το ότι με χρήση π.χ. 32 μόνο *bits* πληροφορίας, μπορούμε να αναπαραστήσουμε ένα πραγ-

ματικό αριθμό, σημαίνει ότι έχουμε σημαντικά οφέλη από άποψη απαιτούμενης πληροφορίας(συμπίεση). Συνοψίζοντας, τα δυο σημαντικά χαρακτηριστικά που εισαγάγει ο κβαντισμός, είναι η συμπίεση από τη μια αλλά και η εισαγωγή θορύβου από την άλλη. Στο σχεδιασμό ενός συστήματος κβαντισμού, το παραπάνω *trade-off* παίζει καθοριστικό ρόλο.

Το ερώτημα που τίθεται από το παραπάνω *trade-off* είναι το εξής : ποιός είναι ο ελάχιστος ρυθμός μετάδοσης για να μπορέσουμε να αναπαριστήσουμε την πηγή πληροφορίας, δεχόμενοι ένα συγκεκριμένο επίπεδο απώλειας. Προκειμένου να απαντήσουμε στο παραπάνω ερώτημα, μπορούμε να χρησιμοποιήσουμε τη θεωρία ρυθμού-απώλειας (*rate-distortion theory*) και την αντίστοιχη συνάρτηση που παρέχει (*rate-distortion function*). Αυτή είναι αρκετά χρήσιμη στον σχεδιασμό συστημάτων κβαντισμού σε θεωρητικό όμως κυρίως επίπεδο. Και αυτό, γιατί εξαρτάται από τον ορισμό της απώλειας, αλλά και υποθέσεις για την κατανομή της πηγής πληροφορίας. Για παράδειγμα, εάν θεωρήσουμε μια πηγή με γκαουσιανή κατανομή μηδενικού μέσου και μεταβλητότητας σ^2, αποδεικνύεται ότι η συνάρτηση ρυθμού-απώλειας, δίδεται από τη σχέση $\frac{1}{2}log\frac{\sigma^2}{D}$ (με απώλεια D τετραγωνικού σφάλματος). Στην πράξη, η υπόθεση για την κατανομή μιας πηγής είναι αρκετά απλοποιημένη, αλλά επίσης δύσκολη είναι και η επιλογή μιας κατάλληλης μετρικής.

Στο παραπάνω παράδειγμα όπου έχουμε απεικόνιση ενός μόνο συμβόλου εξόδου, μιλάμε για βαθμωτό κβαντισμό (*scalar quantization*). Στην περίπτωση που η μονάδα απεικόνισης είναι ένα διάνυσμα συμβόλων εξόδου, έχουμε διανυσματικό κβαντισμό (*vector quantization*). Οι δυο αυτές μεθοδολογίες έχουν μελετηθεί σε βάθος εδώ και χρόνια, και έχουν προταθεί διάφορες τεχνικές για την αποτελεσματικότερη χρήση τους. Εδώ θα περιγράψουμε πολύ σύντομα τις δυο αυτές μεθοδολογίες και θα δούμε πως χρησιμοποιούνται στις αμέσως επόμενες παραγράφους.

2.4 Βαθμωτός κβαντισμός

Στο βαθμωτό κβαντισμό, έχουμε απεικόνιση του συμβόλου εισόδου σε μια μόνο, από ένα σύνολο προκαθορισμένων τιμών που ονομάζονται στάθμες και μπορούμε να τις φανταστούμε σαν προεπιλεγμένα σημεία στον άξονα των πραγματικών αριθμών. Θεωρούμε ότι έχουμε M περιοχές που χωρίζουν τον άξονα των πραγματικών αριθμών. Κάθε τέτοια περιοχή $R_k, 1 \leq k \leq M$ αντιπροσωπεύεται από μια στάθμη $\hat{x}_k$. Αν η τιμή εισόδου x ανήκει στη περιοχή R_k, τότε απεικονίζουμε την είσοδο στην στάθμη της περιοχής. Αυτή η απεικόνιση (συνάρτηση κβαντισμού Q) ορίζεται ως : $Q(x) = \hat{x}_k \quad \forall x \in R_k$. Η συνάρτηση κβαντισμού είναι μη-γραμμική και μη-αναστρέψιμη, αφού όλα τα σημεία στην

R_k απεικονίζονται στην τιμή $\hat{x}_k$. Επειδή η απεικόνιση είναι μη-αναστρέψιμη, κάποια πληροφορία χάνεται για πάντα.

Οι στάθμες αναπαρίστανται ως δυαδική ακολουθία. Αφού έχουμε συνολικά M περιοχές ο ρυθμός πληροφορίας θα είναι $logM$. Υπάρχουν 2 κατηγορίες βαθμωτού κβαντισμού : ομοιόμορφος και μη-ομοιόμορφος κβαντισμός. Στη πρώτη κατηγορία οι περιοχές έχουν ίδιο μήκος ενώ στη δεύτερη όχι. Η δεύτερη επιτρέπει την καλύτερη επιλογή σταθμών, με αποτέλεσμα να είναι πιο αποτελεσματική σε σχέση με τη πρώτη, για ίδιο αριθμό σταθμών. Τα κριτήρια βέλτιστης απόδοσης ενός κβαντιστή είναι γνωστά και ως συνθήκες *Lloyd-Max* και είναι τα εξής :

- Τα όρια των περιοχών είναι τα ισαπέχοντα σημεία από 2 γειτονικές στάθμες. Αυτή η συνθήκη ονομάζεται κανόνας *nearest-neighbour*.
- Οι στάθμες είναι τα *centroids (conditional expected value)* των περιοχών.

Επειδή οι κανόνες αυτοί δεν παρέχουν αναλυτικές λύσεις, η διαδικασία που ακολουθείται είναι το ξεκίνημα με κάποιο σύνολο περιοχών και η εύρεση των *centroids* με βάση το 2ο κριτήριο. Για τα νέα αυτά *centroids*, δημιουργούνται νέες περιοχές με βάση το 1ο κριτήριο. Η διαδικασία συνεχίζεται, εναλλάσσοντας τα κριτήρια, μέχρι η απώλεια να φτάσει κάτω από τα επιθυμητά επίπεδα. Ο επαναληπτικός αυτός αλγόριθμος, αποδεικνύεται ότι συγκλίνει σε κάποιο τοπικό ελάχιστο και οι αρχικές συνθήκες (επιλογή περιοχών), είναι αυτές που καθορίζουν το που θα συγκλίνει ο αλγόριθμος.

2.5 Διανυσματικός κβαντισμός - VQ

Ο διανυσματικός κβαντισμός (VQ από εδώ και πέρα) αποτελεί μια γενίκευση του βαθμωτού (ή και αντίστροφα, ο βαθμωτός είναι ειδική περίπτωση του διανυσματικού). Πηγαίνοντας από την μια διάσταση στη πολυδιάστατη περίπτωση, νέες ιδέες, έννοιες και τεχνικές εμφανίζονται, που επιτρέπουν τη δημιουργία αποτελεσματικών και επιτηδευμένων εφαρμογών συμπίεσης. Ένα διάνυσμα μπορεί να περιγράψει οποιοδήποτε πρότυπο (*pattern*) και από αυτή την άποψη ο VQ μπορεί να θεωρηθεί σαν μια μορφή αναγνώρισης προτύπων, όπου το διάνυσμα εισόδου 'προσεγγίζεται' με ένα προκαθορισμένο από ένα σύνολο τέτοιων πρότυπων διανυσμάτων. Τα πρότυπα αυτά διανύσματα ονομάζονται *codewords* ή και *centroids* και το σύνολο αυτών *codebook*.

Μια μαθηματική αντιστοιχία των παραπάνω εκφράζεται με τα εξής : Θεωρούμε ότι έχουμε M περιοχές στον N-διάστατο χώρο $R_k, 1 \leq k \leq M$ και μονάδα απεικόνισης είναι μπλοκ N συμβόλων εξόδου (ή αλλιώς διάνυσμα μήκους N). Αν $\mathbf{x} \in \mathbf{R}^N$ και $\mathbf{x} \in R_k$ τότε αυτό προσεγγίζεται (κβαντίζεται)

σύμφωνα με την $Q(\mathbf{x}) = \hat{\mathbf{x}}_k$. Έχουμε δηλαδή μια απεικόνιση : $Q : \mathbf{R}^N \rightarrow C$, με $C = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_k, ...\hat{\mathbf{x}}_M)$ και $\hat{\mathbf{x}}_k \in \mathbf{R}^N$. Αφού έχουμε M τέτοιες τιμές κβαντισμού (*codebook* C), απαιτείται ρυθμός μετάδοσης $R = \frac{logM}{N} bits/$σύμβολο εξόδου.

2.5.1 Ανάλυση απόδοσης και πλεονεκτήματα VQ

Σύμφωνα με τις θεμελιώδεις αρχές της θεωρίας 'ρυθμού-απώλειας', αποδεικνύεται ότι με χρήση VQ, μπορεί να επιτευχθεί πάντα καλύτερη απόδοση, από τη περίπτωση χρήσης βαθμωτού κβαντισμού. Στην πραγματικότητα, για πολύ καιρό η θεωρία αυτή είχε περιορισμένη απήχηση, γιατί από τη μια, η θεωρία του *Shannon* δεν παρείχε μεθοδολογίες σχεδιασμού VQ συστημάτων και από την άλλη οι βαθμωτοί κβαντιστές παρείχαν σχετικά ικανοποιητική απόδοση. Στο τέλος της δεκαετίας του '70, με την εισαγωγή του απλού επαναληπτικού αλγορίθμου για σχεδιασμό βαθμωτών κβαντιστών που βασίζεται στα κριτήρια βελτιστοποίησης (όπως είδαμε στην προηγούμενη παράγραφο - *Lloyd*), έγινε αντιληπτό, ότι αυτός είναι εύκολα προσαρμόσιμος στην περίπτωση του VQ, φέρνοντας έτσι στο προσκήνιο το ενδιαφέρον για VQ μεθοδολογίες. Από τότε μέχρι σήμερα υπάρχει έντονο ερευνητικό ενδιαφέρον και έχουν αναπτυχθεί πολλές αποτελεσματικές τεχνικές VQ, βασιζόμενες σε διάφορες παραλλαγές του επαναληπτικού αλγορίθμου, με εφαρμογές σε πολλά είδη πηγών πληροφορίας, όπως ομιλίας και εικόνας.

Είδαμε ότι η συνάρτηση ρυθμού απώλειας εξαρτάται από τη γνώση της κατανομής των συμβόλων εξόδου της πηγής και την επιλογή μιας μετρικής για τον υπολογισμό της απώλειας. Επειδή η κατανομή εξαρτάται από τα χαρακτηριστικά της πηγής, χρησιμοποιούμε για την εκτίμησή της, ένα μεγάλο αριθμό διανυσμάτων εκπαίδευσης (*training set*) που για την περίπτωση που έχουμε πηγή ομιλίας δεν είναι άλλα από τα ακουστικά διανύσματα που παράγει το *front-end*. Οι πιο συχνές επιλογές για την μετρική απώλειας στη περίπτωση πηγών φωνής, είναι η απώλεια τετραγωνικού σφάλματος με ή χωρίς χρήση βαρών και η απώλεια *Itakura-Saito*. Η τελευταία είναι ιδιαίτερα χρήσιμη για ακουστικά διανύσματα που παράγονται από τεχνικές κωδικοποίησης βασισμένες στο μοντέλο παραγωγής φωνής, που θα δούμε σε επόμενη παράγραφο. Στην τεχνική που αναπτύξαμε χρησιμοποιούμε την απώλεια τετραγωνικού σφάλματος με βάρη και συγκεκριμένα την *Mahalanobis* απώλεια που ορίζεται ως εξής :

$$d(\mathbf{x}, \hat{\mathbf{x}}) = (\mathbf{x} - \hat{\mathbf{x}})^T W (\mathbf{x} - \hat{\mathbf{x}}) \tag{2.1}$$

με πίνακα βαρών τον αντίστροφο πίνακα συνδιακύμανσης $W = E[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T]^{-1}$ και $\overline{\mathbf{x}} = E[\mathbf{x}]$.

Ο επαναληπτικός αλγόριθμος που χρησιμοποιούμε, περιγράφεται στη παράγραφο 2.9.2. Στόχος του αλγορίθμου είναι η ελαχιστοποίηση της μέσης απώλειας, δηλαδή του όρου $Ed(\mathbf{x}, \hat{\mathbf{x}})$ που στην πράξη μπορεί να αντικατασταθεί με τον επί μακρόν δειγματοληπτικό μέσο (*long term sample average*) : $lim_{n\to\infty} \sum_{i=0}^{n-1} d(\mathbf{x_i}, \hat{\mathbf{x_i}})$, υπό την προϋπόθεση ότι η ακολουθία είναι εργοδική(*ergodic*) και στάσιμη(*stationary*). Ακόμα όμως και να μην ισχύει αυτή η συνθήκη, στην πράξη είναι αρκετό η πηγή να είναι ασυμπτωτικά στάσιμη στο μέσο (*mean stationary*), κάτι που όντως ισχύει στη περίπτωση της φωνής (*long and short term stationary*).

Είδαμε στην αρχή της παραγράφου, ότι ο ρυθμός μετάδοσης για VQ, είναι $R = \frac{log_2 M}{N} bits/$σύμβολο εξόδου. Αν συγκρίνουμε αυτό το ρυθμό με το ρυθμό στη περίπτωση του βαθμωτού κβαντισμού ($logM$) βλέπουμε ότι ο ίδιος αριθμός *bits* αυτή τη φορά μοιράζεται μεταξύ πολλών συμβόλων εξόδου. Αυτό βέβαια δεν σημαίνει ότι απαιτείται ίδια ποσότητα πληροφορίας στις 2 περιπτώσεις, αλλά ότι στην 2η περίπτωση κατανέμεται σε περισσότερα σύμβολα εξόδου, με μικρότερη πιθανότητα σπατάλησης *bits*. Στην περίπτωση αυτή, ένα σύμβολο εξόδου μπορεί να απαιτεί κλασματικό (*fractional*) ποσό πληροφορίας όπως π.χ. 0.38 *bits* και αυτό δεν είναι το μοναδικό πλεονέκτημα του VQ. Αποδεικνύεται ότι οι τεχνικές VQ δίνουν καλύτερη απόδοση, καθώς εκμεταλλεύονται καλύτερα την ύπαρξη γραμμικής (*correlation*) και μη γραμμικής εξάρτησης μεταξύ των συμβόλων εξόδου, αλλά αποδίδουν καλύτερα ακόμα και στη περίπτωση που αυτή δεν υπάρχει. Τέλος εκμεταλλεύονται καλύτερα την κατανομή της πηγής και επιτρέπουν τον σχηματισμό περιοχών με σχήμα χωρίς περιορισμούς στον N-διάστατο χώρο.

2.5.2 Είδη VQ, παράμετροι σχεδιασμού και τεχνικές

Οι τεχνικές VQ χωρίζονται σε 2 κατηγορίες : κβαντιστές με μνήμη και χωρίς μνήμη. Οι δεύτεροι αποτελούν μια εναλλακτική λύση σε περίπτωση που οι πρώτοι έχουν μεγάλες απαιτήσεις σε πόρους, όπως μνήμη και υπολογιστική ισχύς (μεγάλα διανύσματα και μεγάλα *codebooks*) και εκμεταλλεύονται το γεγονός ύπαρξης συσχέτισης μεταξύ διαδοχικών διανυσμάτων. Όμως με προσεκτικό σχεδιασμό των πρώτων, μπορούμε να έχουμε αρκετά αποδοτικό κβαντισμό, καθιστώντας τη χρήση των δεύτερων μη υποχρεωτική.

Ο αριθμός τεχνικών VQ είναι αρκετά μεγάλος για κβαντιστές χωρίς μνήμη και ακόμα μεγαλύτερος για αυτούς με μνήμη. Επίσης υπάρχουν παράμετροι που μπορούν να επηρεάσουν τον τρόπο σχεδιασμού για αυτές τις τεχνικές. Με αυτή τη πληθώρα διαφορετικών προσεγγίσεων, κανείς πρέπει να διαλέξει προσεκτικά την κατάλληλη, προκειμένου να έχει τα μέγιστα ωφέλη. Στη συνέχεια, θα επικεντρωθούμε σε κάποιες τέτοιες παραμέτρους και τεχνικές που επιλέ-

χθηκαν από την κωδικοποίηση που κάναμε για αυτή την εργασία, με δεδομένη την επιλογή κβαντιστή χωρίς μνήμη.

Επιλογή αρχικού codebook

Μια τέτοια παράμετρος, είναι το μέγεθος του αρχικού *codebook* (δηλαδή ο αριθμός *centroids* σε αυτό) που θα χρησιμοποιήσουμε στον επαναληπτικό αλγόριθμο. Δυο προσεγγίσεις υπάρχουν : στην πρώτη, το μέγεθος είναι όσο και το τελικό (το οποίο καθορίζουμε με τον αριθμό των *bits* που έχουμε επιλέξει να διαθέσουμε) και τα αρχικά *centroids* επιλέγονται π.χ. τυχαία από το σετ εκπαίδευσης. Στη δεύτερη το μέγεθος είναι μικρό (π.χ. ένα : ο μέσος όρος όλων των υποδιανυσμάτων στο σετ εκπαίδευσης) και διπλασιάζεται από επανάληψη σε επανάληψη του αλγορίθμου, μέσα από μια διαδικασία διαχωρισμού (*splitting*) των παρόντων *centroids*, μέχρι την ικανοποίηση κάποιου κριτηρίου (π.χ. επίπεδο απώλειας, επίδοση αναγνώρισης). Η απόφαση για την παράμετρο αυτή, εξαρτάται από την παραλλαγή του αλγορίθμου που χρησιμοποιείται και εμείς επιλέξαμε την δεύτερη.

Product VQ

Μια άλλη επιλογή που έχουμε, είναι να χωρίσουμε το αρχικό διάνυσμα σε μικρότερα και να εφαρμόσουμε *VQ* για καθένα από αυτά. Αυτό προκύπτει, γιατί πολλές φορές μπορούμε να διακρίνουμε τέτοια μικρότερα γκρουπ μέσα στο διάνυσμα, βάσει κάποιων κοινών χαρακτηριστικών. Στη περίπτωση της φωνής, η μέθοδος αυτή επιλέγεται κυρίως για ακουστικά διανύσματα που παράγονται από *model-based* τεχνικές (βλέπε 2.8). Για *front-ends* όπως αυτά που είδαμε στο Κεφάλαιο 1, ο τρόπος αυτός χρησιμοποιείται σχετικά σπάνια, με πιο κοινό χαρακτηριστικό διαχωρισμού, την ποικιλία τιμών (*dynamic range*) των στοιχείων του ακουστικού διανύσματος. Σύμφωνα με το τελευταίο, ο πιο εμφανής διαχωρισμός είναι ανάμεσα στην ενέργεια και τις υπόλοιπες 12 *cepstral* παραμέτρους.

Το βασικό πλεονέκτημα του διαχωρισμού σε μικρότερα διανύσματα, είναι η μείωση του κόστους αποθήκευσης των *codebooks* και του υπολογισμού που απαιτείται για τον υπολογισμό της απώλειας, αφού τώρα το διάνυσμα είναι μικρότερο. Το μειονέκτημα είναι ότι αυτός ο διαχωρισμός εμποδίζει τον κβαντιστή να εκμεταλλευτεί π.χ. την συσχέτιση που υπάρχει μεταξύ στοιχείων, που τώρα βρίσκονται σε διαφορετικά διανύσματα (σύμφωνα με τη θεωρία, όσο μεγαλύτερο το διάνυσμα, τόσο μεγαλύτερα τα ωφέλη από άποψη συμπίεσης). Φαίνεται ότι στη πράξη, προηγούμενες σχεδιάσεις *VQ* απέφευγαν τη χρήση αυτής της μεθόδου, παρασυρόμενοι από το παραπάνω μειονέκτημα, ή από το

γεγονός ότι ο διαμοιρασμός των διανυσμάτων γινόταν με λάθος κριτήρια, δίνοντας μη ικανοποιητικά αποτελέσματα έως τώρα.

Εμείς επιμείναμε σε αυτή τη μέθοδο, και καταφέραμε να την εκμεταλλευτούμε όσο περισσότερο γινόταν. Μάλιστα τα τελικά εντυπωσιακά αποτελέσματα της κωδικοποίησης που χρησιμοποιήσαμε, δείχνουν ότι αυτή η μέθοδος μπορεί να βελτιώσει πολύ ένα σχήμα κωδικοποίησης, με το μειονέκτημά της να επηρεάζει ελάχιστα, σε σχέση με τα ωφέλη που μπορεί να δημιουργήσει, αρκεί να χρησιμοποιηθεί κατάλληλα. Σχεδιάζοντας ένα κβαντιστή, πρέπει κανείς να σταθμίσει σωστά τα συν και τα πλην. Για παράδειγμα, η χρήση ολόκληρου του διανύσματος μπορεί θεωρητικά να είναι η καλύτερη επιλογή, πρακτικά όμως δημιουργεί πολύ μεγάλες απαιτήσεις σε μνήμη και υπολογιστική ισχύ. Προκειμένου να δούμε τα ωφέλη της μεθόδου, θα πρέπει να επιλεχθούν κατάλληλα τα μικρότερα διανύσματα (υποδιανύσματα) αλλά και οι συντελεστές που τα αποτελούν. Στο παράδειγμα που δώσαμε παραπάνω (12 συντελεστές, ενέργεια) τα ωφέλη είναι πολύ μικρά, ίσως γιατί το κριτήριο που επιλέχθηκε δεν είναι τόσο μεγάλης σημασίας.

Ενας καλύτερος διαχωρισμός είναι αυτός που θα μοιράσει το διάνυσμα των 13 στοιχείων σε ένα μεγαλύτερο αριθμό υποδιανυσμάτων (π.χ. 5) καθένα από τα οποία απαιτεί περίπου ίδιο αριθμό *bits*. Ας υποθέσουμε ότι οι δυο μέθοδοι απαιτούν τον ίδιο αριθμό *bits* (αφού αντιπροσωπεύουν το ίδιο διάνυσμα με 2 διαφορετικές μορφές), έστω 20 και ότι στη δεύτερη περίπτωση τα 5 διανύσματα απαιτούν 5, 5, 4, 4 και 2 *bits* αντίστοιχα. Χρησιμοποιώντας την πρώτη προσέγγιση απαιτούνται 2^{20}, δηλαδή πάνω από ένα εκατομμύριο *centroids* στο μοναδικό *codebook*, ενώ στη δεύτερη 5 *codebooks* με $2^5 + 2^5 + 2^4 + 2^4 + 2^2$, δηλαδή 32, 32, 16, 16 και 4 *centroids* αντίστοιχα (σύνολο 100). Τα ωφέλη είναι απίστευτα, από άποψη χρόνου κβαντισμού και αποθήκευσης, ακόμα και αν έχουμε μικρή απώλεια ως προς τη βέλτιστη θεωρητική συμπίεση (δηλαδή για την ίδια συμπίεση η πρώτη μέθοδος να απαιτούσε τελικά 18 ή ακόμα και 15 *bits*, κάτι που δεν αλλάζει όμως την εικόνα).

Τεχνικές ψαξίματος

Τέλος, η τελευταία επιλογή που θα εξετάσουμε, είναι ο τρόπος που είναι δομημένα τα *centroids* μέσα στα *codebooks*, ο οποίος καθορίζει και τον αριθμό των συγκρίσεων που απαιτούνται για να βρεθεί το 'πλησιέστερο' στο διάνυσμα εισόδου *centroid* (*nearest-neighbour search*). Ο πιο απλός τρόπος, είναι να χρησιμοποιηθεί πλήρες ψάξιμο, δηλαδή όλα τα *centroids*. Αν και εξαιρετικά απλός, για μεγάλα *codebooks* αυτός ο τρόπος είναι πολύ αργός. Η άλλη επιλογή είναι να δομηθούν τα *centroids* με ένα ιεραρχικό τρόπο (π.χ. δυαδικό δέντρο). Με αυτό τον τρόπο, για M *centroids* απαιτούνται μόλις *logM* συγκρίσεις αντί

για M που απαιτούνται στην προηγούμενη περίπτωση. Αν και πολύ γρηγορότερος, αυτός ο τρόπος απαιτεί αλλαγές στον αλγόριθμο εκπαίδευσης, αλλά είναι και υποβέλτιστος σε σχέση με τον πρώτο, αφού τα διανύσματα που χρησιμοποιούνται για την εκπαίδευση των *centroid* σε κάθε επανάληψη, δεν είναι το σύνολο αυτών του σετ εκπαίδευσης, αλλά ένα όλο και μικρότερο γκρουπ αυτών, τα οποία σε προηγούμενη επανάληψη ομαδοποιήθηκαν στην περιοχή του παρόντος *centroid*. Με δεδομένη την χρήση υποδιανυσμάτων, τα *codebooks* είναι αρκετά μικρά, άρα δεν έχει τόσο μεγάλη σημασία η χρήση της δεύτερης μεθόδου αφού επιπλέον είναι και μη βέλτιστη.

2.6 Τεχνικές κωδικοποίησης φωνής

Οι τεχνικές κωδικοποίησης χωρίζονται σε 3 κατηγορίες :

- *Waveform*
- *Model-based*
- *Hybrid*

Σε κάθε μια από τις παραπάνω κατηγορίες συναντάμε τεχνικές που παρέχουν διαφορετικούς ρυθμούς μετάδοσης, ακρίβεια αναπαράστασης της πληροφορίας, απόδοση παρουσία θορύβου και πολυπλοκότητα υλοποίησης. Κάθε κατηγορία εξυπηρετεί λοιπόν διαφορετικές ανάγκες.

2.7 Waveform Coding τεχνικές

Οι τεχνικές αυτής της κατηγορίας, είναι στη πλειοψηφία τους σχετικά απλές. Σκοπός είναι η πιστή αναπαραγωγή του σήματος της φωνής στο δέκτη. Κωδικοποιητές (*coders*) κυματομορφής έχουν σχεδιαστεί τόσο στο πεδίο του χρόνου όσο και στο πεδίο της συχνότητας και είναι αρκετά γρήγοροι λόγω ακριβώς της σχετικά απλής λειτουργίας τους.

2.7.1 Τεχνικές στο πεδίο του χρόνου

Εισάγουν από ελάχιστες έως καθόλου υποθέσεις για το σήμα που κωδικοποιείται και χρησιμοποιούνται σε υψηλούς σχετικά ρυθμούς μετάδοσης. Κάνουν χρήση βαθμωτού κβαντισμού συνήθως και μερικά τέτοια παραδείγματα των τεχνικών αυτών ακολουθούν ευθύς αμέσως :

Pulse Code Modulation (PCM)

Αποτελεί την πιο απλή μέθοδο. Το σήμα δειγματοληπτείται στη συχνότητα *Nyquist* και κάθε δείγμα κβαντίζεται στην κοντινότερη στάθμη και στη συνέχεια μεταδίδεται σαν δυαδικός αριθμός. Όσα περισσότερα *bits* διαθέσουμε, τόσες περισσότερες στάθμες έχουμε, με αποτέλεσμα να έχουμε πιο πιστή αναπαράσταση του σήματος. Το σήμα της φωνής χαρακτηρίζεται από το γεγονός ότι μικρά πλάτη (*amplitudes*) συμβαίνουν πολύ συχνότερα από τα μεγάλα. Μια βελτίωση είναι επομένως, οι στάθμες να απέχουν λιγότερο μεταξύ τους στα μικρά πλάτη και περισσότερο στα μεγάλα. Αυτό οδηγεί στη χρήση μη ομοιόμορφου βαθμωτού κβαντιστή, ο οποίος συμπιέζει το σήμα χρησιμοποιώντας κάποια λογαριθμική συνάρτηση και στη συνέχεια το κβαντίζει χρησιμοποιώντας ομοιόμορφο κβαντιστή. Τέτοιοι λογαριθμικοί συμπιεστές χρησιμοποιούνται για τη μετάδοση φωνής στα τηλεφωνικά δίκτυα και είναι γνωστοί σαν law & *A-law compressors*.

Differential PCM (DPCM)

Μια πρώτη βελτίωση στην *PCM* μέθοδο είναι ο κβαντισμός να γίνεται στην διαφορά μεταξύ 2 διαφορετικών δειγμάτων. Αυτή είναι μια σημαντική βελτίωση, αφού στο σήμα της φωνής, διαδοχικά δείγματα έχουν μεγάλο βαθμό συσχέτισης και επομένως η διαφορά μεταξύ 2 διαφορετικών δειγμάτων είναι μικρή. Μια βασική επέκταση της παραπάνω ιδέας είναι να κάνουμε κάποια πρόβλεψη για το τρέχον δείγμα, και να κβαντίζουμε τη διαφορά μεταξύ του πραγματικού δείγματος και της προβλεπόμενης τιμής του. Έτσι περιμένουμε να μειώσουμε ακόμα περισσότερο το απαιτούμενο ρυθμό δεδομένων.
Για την πρόβλεψη του επόμενου δείγματος χρησιμοποιείται ένας γραμμικός προβλέπτης της μορφής : $\hat{x}_n = \sum_{i=1}^{p} a_i x_{n\ i}$,όπου p είναι τα προηγούμενα δείγματα που χρησιμοποιούμε και a_i είναι το βάρος για το αντίστοιχο προηγούμενο δείγμα.

Adaptive DPCM (ADPCM)

Οι παραπάνω τεχνικές βασίζονται στην υπόθεση ότι το σήμα εισόδου είναι στάσιμο (*stationary*), δηλαδή οι στατιστικές του ιδιότητες (π.χ. μεταβλητότητα, αυτοσυσχέτιση) είναι σταθερές με το χρόνο. Στην περίπτωση του σήματος της φωνής οι παραπάνω ιδιότητες μεταβάλλονται αργά με το χρόνο (*quasi-stationary*). Για να εκμεταλλευτούμε και αυτό το χαρακτηριστικό του σήματος της φωνής οι κωδικοποιητές πρέπει να μπορούν να προσαρμοστούν στις χρονικά μεταβαλλόμενες αλλαγές των στατιστικών μεταβολών του σήματος. Έτσι έχουν αναπτυχθεί προσαρμοστικές εκδόσεις του *DPCM*, που

παραμέτρων που αντιστοιχούν στις χρονικά μεταβαλλόμενες παραμέτρους του φωνητικού σωλήνα.

Το μοντέλο φαίνεται στο σχήμα 2.1. Όπως φαίνεται, υπάρχουν 2 δυνατές πηγές. Η πρώτη πηγή αντιστοιχεί στον ήχο χωρίς φωνή και η δεύτερη σε αυτόν με φωνή (*unvoiced & voiced sounds*). Στην 2η πηγή η φωνή παράγεται όταν οι φωνητικές χορδές πάλλονται σε κάποια βασική συχνότητα f_0. Μια πηγή κάθε χρονική στιγμή διεγείρει το φίλτρο που αναπαριστάνει το φωνητικό σωλήνα και έτσι παράγεται το σήμα της φωνής. Η παραγωγή των παραμέτρων του φωνητικού σωλήνα γίνεται με χρήση γραμμικής πρόβλεψης (*linear predictive coding*). Το σύνολο των παραμέτρων που προκύπτουν περιλαμβάνουν τους συντελεστές πρόβλεψης, το κέρδος (*gain*) του σήματος, το είδος εισόδου που έχουμε και τη συχνότητα f_0 , εφόσον η είσοδος είναι *voiced*. Οι παραπάνω παράμετροι μεταδίδονται στον δέκτη όπου χρησιμοποιώντας το ίδιο μοντέλο, ανακατασκευάζεται το αρχικό σήμα (για αυτό το λόγο οι τεχνικές αυτές ονομάζονται επίσης *analysis by synthesis* τεχνικές). Για κάθε 20-30 *msecs* που το σήμα της φωνής μπορεί να θεωρηθεί *stationary* υπολογίζονται οι συντελεστές του φίλτρου σύμφωνα με την εξίσωση :

$$X_n = \sum_{i=1}^{p} aiX_{n-1} + Gw_n \tag{2.2}$$

όπου G είναι το κέρδος, wn η ακολουθία εισόδου, a_i οι συντελεστές του φίλτρου και p ο αριθμός πόλων του φίλτρου.

Από αυτές τις παραμέτρους και με κατάλληλους μετασχηματισμούς μπορούν να προκύψουν ακουστικά διανύσματα ισοδύναμα με αυτά που είδαμε στο Κεφάλαιο 1, τα οποία βέβαια μπορούν επίσης να χρησιμοποιηθούν για αναγνώριση. Άρα η μέθοδος αυτή αποτελεί ουσιαστικά ένα εναλλακτικό *front-end* μηχανισμό. Αποτελεί φυσικά και μέθοδο κωδικοποίησης φωνής, αφού από το αρχικό σήμα φωνής έχει ήδη προκύψει μια ακολουθία ακουστικών διανυσμάτων.

Βασικό χαρακτηριστικό των *vocoders* είναι οι χαμηλοί ρυθμοί μετάδοσης αλλά και η χαμηλή ποιότητα. Υπάρχουν πολλές υλοποιήσεις *vocoders* με πιο διαδεδομένους όσους στηρίζονται σε τεχνικές γραμμικής πρόβλεψης, σαν αυτή που μόλις περιγράφτηκε. Οι υβριδικοί κωδικοποιητές (επίσης *vocoders*), προσπαθούν να χρησιμοποιήσουν μια πιο σύνθετη διέγερση από τους *LPC vocoders*, καταφέρνοντας έτσι καλή ποιότητα για χαμηλούς-μέσους ρυθμούς μετάδοσης. Υλοποιήσεις *vocoders* (υβριδικών και μη) είναι οι *channel, phase, cepstral & formant vocoders*, καθώς και οι *RELP, CELP & VSELP vocoders*. Μάλιστα η κωδικοποίηση GSM στη κινητή τηλεφωνία δευτέρας γενεάς, βασίζεται σε *RPE-LTP* υβριδικό κωδικοποιητή, ενώ οι *CELP* αποτελούν το πρότυπο για κινητή τηλεφωνία στην Βόρειο Αμερική.

2.9 Το σχήμα κωδικοποίησης που χρησιμοποιήσαμε

Στις προηγούμενες παραγράφους είδαμε αναλυτικά τεχνικές κβαντισμού και κωδικοποίησης της φωνής. Σημειώθηκε ότι η χρήση του *Filterbank Spectrum Analyzer* σαν μηχανισμού *front-end*, καθώς και η χρήση διανυσματικού κβαντισμού βάσει μικρότερων υποδιανυσμάτων (*product VQ*), παίζουν σημαντικό ρόλο στην επιτυχία του σχήματος κωδικοποίησης αυτής της εργασίας, την οποία και παρουσιάζουμε σε αυτό το δεύτερο μέρος.

Ο *front-end* μηχανισμός παράγει τα ακουστικά διανύσματα με τους 13 *MFCC* συντελεστές για κάθε *frame* (δηλαδή διάστημα 10 *msecs*), με αποτέλεσμα ο ρυθμός πληροφορίας να είναι 1300 συντελεστές αντίστοιχα ανά *second*. Αφού κάθε συντελεστής αντιπροσωπεύεται από μια *float* μεταβλητή απαιτούνται 1300 * 32 *bits/sec*, δηλαδή 41,6 *kbps* αντίστοιχα. Αυτός ο ρυθμός είναι μικρότερος σε σχέση με αυτό της αρχικής πηγής πληροφορίας (δειγματοληπτημένη κυματομορφή σε 8 ή $16Khz$ και με $16bits$ ανά σύμβολο) και σκοπός μας είναι να τον μειώσουμε ακόμα περισσότερο, χρησιμοποιώντας τεχνικές κβαντισμού.

Θυμίζουμε ότι με τον κβαντισμό, ουσιαστικά περιορίζουμε τον αριθμό συμβόλων εξόδου, σε ένα μικρό σύνολο προτύπων συμβόλων εξόδου, καταφέρνοντας να συμπιέσουμε από τη μια το σήμα αλλά και δυστυχώς να εισάγουμε θόρυβο από την άλλη. Στην εισαγωγή αυτού του κεφαλαίου, τονίσαμε ότι επειδή στόχος μας δεν είναι η όσο πιο πιστή αναπαραγωγή του σήματος, αλλά η καλύτερη δυνατή αναγνώριση, την απώλεια πληροφορίας που έχουμε με τον κβαντισμό, τη μετράμε με βάση τα αποτελέσματα της αναγνώρισης. Αυτό μας βοηθάει πρακτικά στο να αξιολογήσουμε τα διάφορα σχήματα κωδικοποίησης που θα αναπτύξουμε. Τα σημαντικότερα αποτελέσματα από τα πειράματα που διεξάχθηκαν προκειμένου να καταλήξουμε στο καλύτερο μέχρι στιγμής αποτέλεσμα, με ρυθμό μετάδοσης μόλις $2kbps$, παρατίθενται στο τέλος του κεφαλαίου. Στο Κεφάλαιο 4, παραθέτουμε κάποια άλλα ενδεικτικά πειράματα που έγιναν, πριν καταλήξουμε στα αποτελέσματα που παρουσιάζουμε σε αυτό το κεφάλαιο.

Σε αυτή την ενότητα, θα θυμίσουμε κάποιες βασικές έννοιες-ορολογίες, όπως αυτές των *codebooks, centroids* και θα ορίσουμε κάποιες άλλες. Στη συνέχεια θα εξετάσουμε τις διαδικασίες εκπαίδευσης (δημιουργία *codebooks* με βάση τον επαναληπτικό αλγόριθμο *k-means*) και κβαντισμού, στην περίπτωση της (*product VQ*) τεχνικής, η οποία αποτελεί και την 'καρδιά' της κωδικοποίησης. Τα παραπάνω πλαισιώνονται με αρκετά σχήματα, προκειμένου να γίνουν ακόμα πιο ξεκάθαρες οι παραπάνω έννοιες και διαδικασίες που περιγράφονται.

2.9.1 Βασικές έννοιες

Εδώ θα θυμίσουμε κάποιες βασικές έννοιες και θα ορίσουμε κάποιες νέες, όπως αυτές θα χρησιμοποιούνται στη συνέχεια, προσαρμοσμένες στην περίπτωση του *VQ* με υποδιανύσματα.

Είσοδος συστήματος κβαντισμού.

Η είσοδος στο σύστημα κβαντισμού είναι το *MFCC* διάνυσμα, όχι όμως απαραίτητα με τη μορφή μονάχα του ενός διανύσματος. Μπορούμε να το αναπαραστήσουμε με περισσότερα (υπο-)διανύσματα, αρκεί αυτά συνδυαζόμενα να μας δίνουν την το αρχικό διάνυσμα. Έτσι, αν υποθέσουμε ότι το *MFCC* διάνυσμα έχει μέγεθος 13 (*high quality* - χωρίς χρήση των παραγώγων), τότε έχουμε επιλογές όπως :

1. Να χρησιμοποιήσουμε το διάνυσμα ως έχει : [1 2 3 ... 13]
2. Να χρησιμοποιήσουμε 13 βαθμωτούς : 1, 2, 3, ..., 13
3. Να χρησιμοποιήσουμε $L(1 \leq L \leq 13)$ υποδιανύσματα, όπως π.χ. : [1 2 ... 7], [8 ... 12], [13] ή [1 3 5 7 ... 13] [2 4 6 ... 12] κ.λ.π.
4. Οποιαδήποτε άλλη επιλογή κριθεί κατάλληλη. Θα δούμε στη συνέχεια το τρόπο επιλογής υποδιανυσμάτων

Και οι 3 παραπάνω επιλογές είναι περιπτώσεις του διανυσματικού κβαντισμού (*Vector quantization*). Η 2η είναι περίπτωση βαθμωτού κβαντισμού, ενώ η 3η είναι ο *code product* κβαντισμός. Στην 1η περίπτωση χρησιμοποιούμε ένα μόνο κβαντιστή, στη 2η δεκατρείς & στη 3η τρεις και δυο κβαντιστές αντίστοιχα.

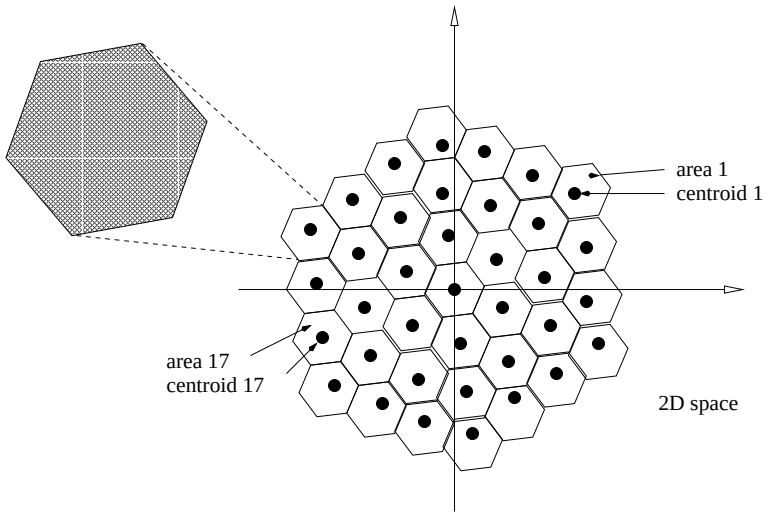
Αναπαράσταση της εξόδου του συστήματος κβαντισμού.

Αφού εξετάσαμε τις δυνατές εισόδους, ας κάνουμε το ίδιο για την έξοδο. Πρώτα όμως να θυμίσουμε ότι στη διαδικασία κβαντισμού απεικονίζουμε ένα δείγμα εισόδου σε ένα μόνο δείγμα, από ένα πεπερασμένο αριθμό δειγμάτων εξόδου (έστω M), τα οποία προκύπτουν κατάλληλα από την διαδικασία εκπαίδευσης των δεδομένων εκπαίδευσης. Το γεγονός ότι τα δείγματα εξόδου είναι πεπερασμένα, μας δίνει ένα επιπλέον τρόπο απεικόνισης. Έτσι έχουμε 2 τρόπους απεικόνισης, αν κβαντίσουμε το δείγμα εισόδου στο i-οστό από τα πεπερασμένα δείγματα :
το i-οστό διάνυσμα (*codeword*) ή απλά τον αριθμό i *(index)*. Το είδος της απεικόνισης είναι αυτό που καθορίζει και το είδος ακουστικών μοντέλων στον αναγνωριστή. Στην 1η περίπτωση όπου απεικονίζουμε ένα δείγμα-διάνυσμα μεγέθους N σε ένα επίσης διάνυσμα μεγέθους N

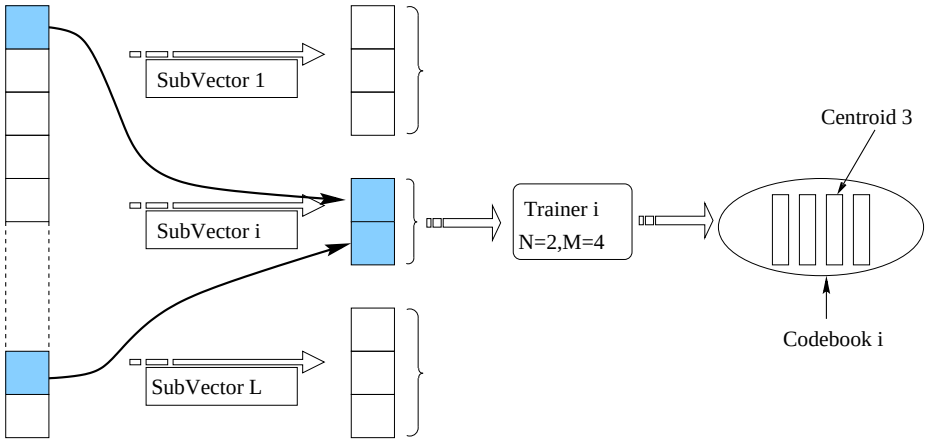
μιλάμε για *Centroid Coded* κβαντισμό. Στη 2η περίπτωση, όπου απεικονίζουμε το δείγμα-διάνυσμα μεγέθους N στον ακέραιο που χαρακτηρίζει το i-οστό αυτό διάνυσμα (δηλ. το i) μιλάμε για *Index Coded* κβαντισμό.

Centroid και *Codebook*

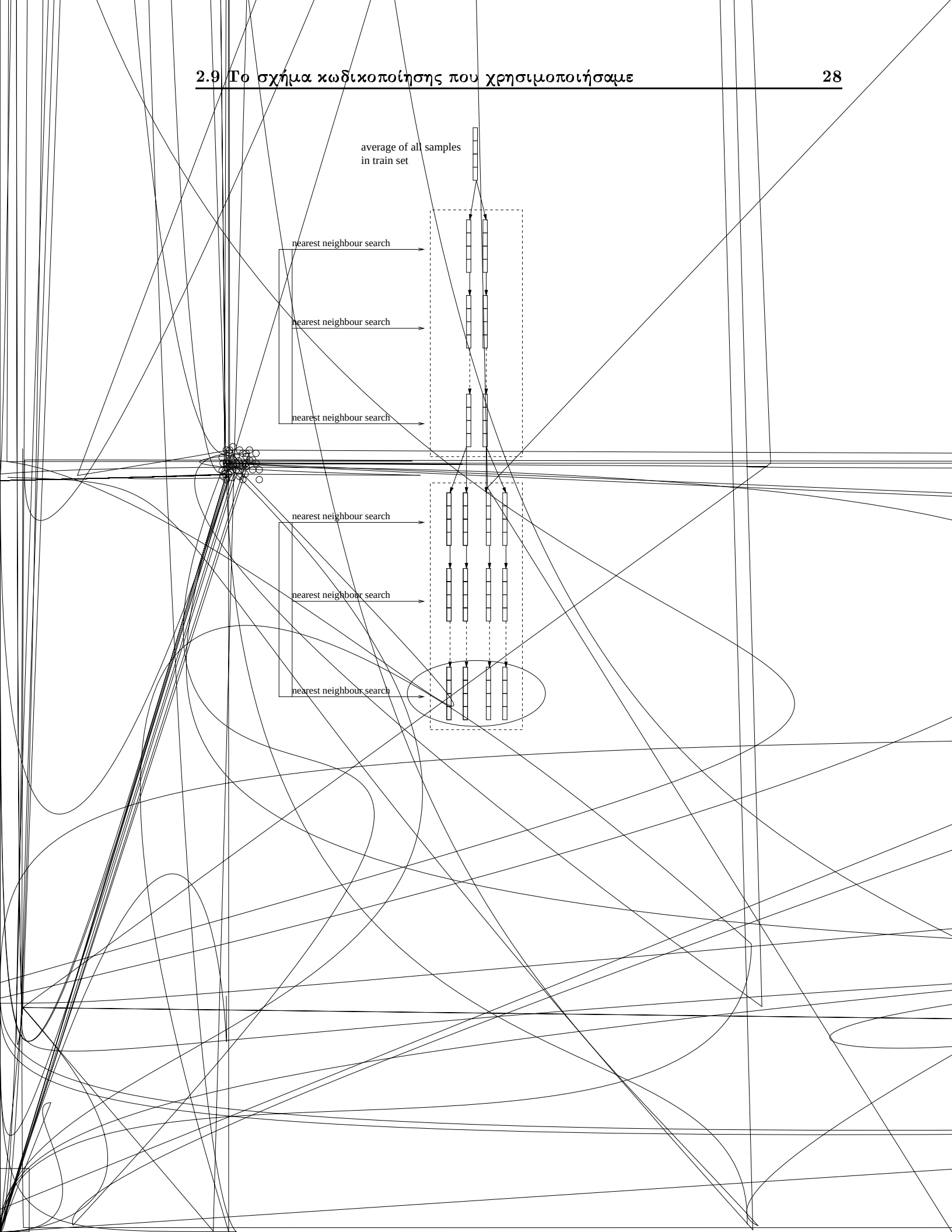
Χωρίζοντας τον N-διάστατο χώρο σε M περιοχές, κάθε μια από αυτές τις περιοχές έχει ένα σημείο αναφοράς, το



Σχήμα 2.2: **Codebook & Centroids.** Παράδειγμα για *N=2* και *M=36* όπου φαίνεται ο διδιάστατος χώρος με 36 περιοχές. Κάθε περιοχή έχει κι ένα *centroid*, το οποίο έχει την ιδιότητα να αντιπροσωπεύει καλύτερα όλα τα σημεία της περιοχής. Το σύνολο των 36 αυτών *centroids* αποτελεί το *codebook* δηλαδή τα 36 διδιάστατα διανύσματα



Σχήμα 2.3: **Διαδικασία εκπαίδευσης.** Χωρίζουμε κάθε *MFCC* διάνυσμα σε L υποδιανύσματα με μέγεθος N_i το καθένα. Για κάθε υποδιάνυσμα, αποφασίζουμε πόσους και ποιους συντελεστές θα περιέχει, καθώς και πόσα *centroids* στο *codebook* (στη συγκεκριμένη περίπτωση 4).


average of all samples
in train set
nearest neighbour search
nearest neighbour search
nearest neighbour search
nearest neighbour search
nearest neighbour search
nearest neighbour search

μός των *bits* που θέλουμε να χρησιμοποιήσουμε. Αν π.χ. έχουμε $B=5$, αυτό σημαίνει ότι τελικά θα έχουμε 32 *centroids* στο *codebook*. Ο αλγόριθμος εκπαίδευσης που χρησιμοποιούμε είναι ο *k-means* (σε κάποιες παραλλαγές είναι γνωστός και ως *generalized Lloyd, LBG* ή και *ISODATA* αλγόριθμος), εφαρμόζεται για κάθε ένα από τα υποδιανύσματα που έχουμε ορίσει και δίνεται παρακάτω :

Αρχικοποίηση (Initialization):
Θέσε αρχικό μοναδικό *centroid*, το υποδιάνυσμα που αντιστοιχεί στη μέση τιμή όλων των τιμών του υποδιανύσματος στο *train set*(σετ εκπαίδευσης). Η επιλογή αυτή είναι αρκετά αντιπροσωπευτική και είμαστε τώρα έτοιμοι να περάσουμε στον επαναληπτικό αλγόριθμο.

Σε κάθε βήμα $i : (0 \leq i < B)$
Βήμα 1 : Χώρισε κάθε ένα από τα n *centroids* $(1 \leq n \leq 2^i)$ σε 2 νέα, εφαρμόζοντας δυαδικό χωρισμό (*binary splitting*).
Αυτό σημαίνει ότι από κάθε *centroid* y_n παίρνουμε 2 νέα, τα : $(y_n^+ = y_n + e)$ & $(y_n^- = y_n - e)$ τα οποία όμως δεν είναι αρκετά αντιπροσωπευτικά και χρειάζονται εκπαίδευση για να γίνουν. Η τιμή του e είναι της τάξης του *0.05*
Βήμα 2 : Για κάθε επανάληψη j (Ο αριθμός των επαναλήψεων αυξάνει γραμμικά με το βήμα : π.χ. $j=i+5$)

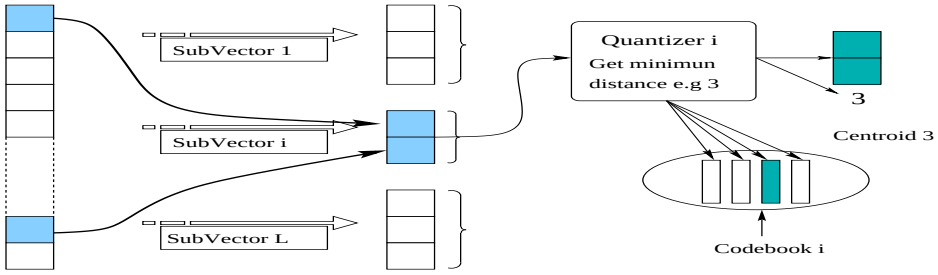
1. Για κάθε υποδιάνυσμα του σετ εκπαίδευσης (*Clustering - Nearest Neighbour search*):
   Υπολόγισε την απόστασή του από όλα τα υπάρχοντα *centroids* και ομαδοποίησέ το στο *centroid* με το οποίο το μέτρο της μεταξύ τους απόστασης είναι ελάχιστο. Το μέτρο της απόστασης που χρησιμοποιείται παίζει μεγάλο ρόλο. Συνήθως χρησιμοποιείται η *Mahalanobis* απόσταση.

2. Υπολόγισε τις νέες τιμές των *centroids (Reestimation - Centroid Update)*:
   Η νέα τιμή για κάθε *centroid*, είναι η μέση τιμή όλων των υποδιανυσμάτων που έχουν ομαδοποιηθεί στην προηγούμενη τιμή του *centroid*

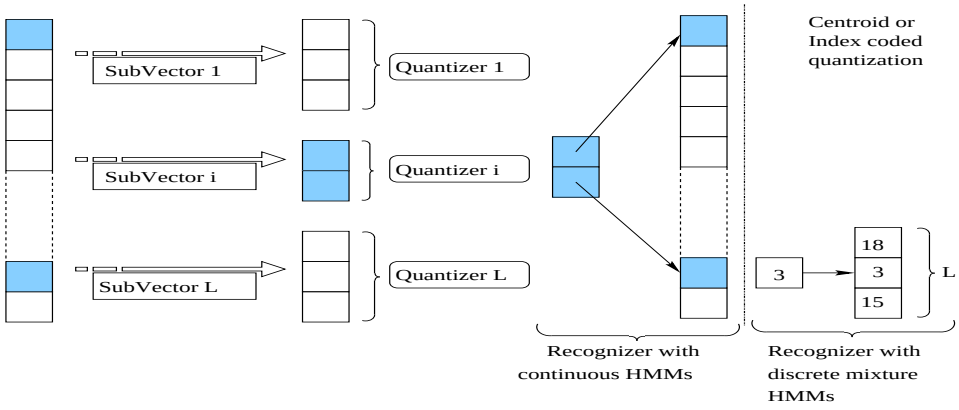
Ο αλγόριθμος φαίνεται καλύτερα στο σχήμα 2.4

2.9.3 Διαδικασία κβαντισμού

Η διαδικασία κβαντισμού είναι πιο απλή. Η ιδέα στην οποία βασίζεται, είναι ότι κάθε υποδιάνυσμα θα πρέπει να κβαντιστεί στο *centroid* εκείνο του *codebook*, με το οποίο μοιάζει περισσότερο ή αλλιώς με το *centroid* με το οποίο η μεταξύ τους απόσταση είναι μικρότερη. Περισσότερα σχόλια στα σχήματα 2.5 και 2.6.



Σχήμα 2.5: **Διαδικασία κβαντισμού.** Για κάθε υποδιάνυσμα i, ο κβαντιστής βρίσκει το *centroid* που το αντιπροσωπεύει καλύτερα, δηλαδή το *centroid* με το οποίο η απόσταση είναι μικρότερη. Όσο για την έξοδο του κβαντιστή, αυτή μπορεί να είναι ή το ίδιο το *centroid* ή το *index* του. Στη συγκεκριμένη περίπτωση, το υποδιάνυσμα i λέμε ότι κβαντίστηκε στο 3ο *centroid*.



Σχήμα 2.6: **Έξοδος συστήματος κβαντιστών και αναγνώριση.** Αν έχουμε *centroid coded* κβαντισμό, από το σύστημα των κβαντιστών προκύπτει ένα ανακατασκευασμένο *MFCC* διάνυσμα, αφού τα *elements* κάθε *centroid* παίρνουν την θέση τους στο αρχικό διάνυσμα. Η αναγνώριση για *centroid coded* κβαντισμό, γίνεται από ένα *Recognizer* με συνεχή *HMM* μοντέλα. Αν έχουμε *index coded* κβαντισμό, η έξοδος των κβαντιστών είναι ένα διάνυσμα από τα αντίστοιχα των *centroid indices*. Προκύπτει έτσι ένα διάνυσμα μήκους L, όσο και ο αριθμός των υποδιανυσμάτων. Η αναγνώριση για *index coded* κβαντισμό γίνεται από ένα *Recognizer* με μείγματα διακριτών *HMMs*.

2.10 Αξιολόγηση κωδικοποίησης

Στις δυο προηγούμενες παραγράφους μιλήσαμε για την διαδικασία εκπαίδευσης και κβαντισμού. Επισημάνθηκε επίσης προηγούμενα ότι η κωδικοποίηση που χρησιμοποιούμε δεν έχει στόχο την ακριβή ανακατασκευή του σήματος της φωνής (ακουστικών διανυσμάτων), αλλά την καλύτερη δυνατή αναγνώριση (κωδικοποίηση φωνής για αναγνώριση). Εύλογα λοιπόν η αξιολόγηση ενός σχήματος κωδικοποίησης γίνεται με την αναγνώριση των κβαντισμένων ακουστικών διανυσμάτων.

2.11 Συμπεράσματα

- Οι τεχνικές κωδικοποίησης χωρίζονται σε δυο κατηγορίες ανάλογα με το αν τηρούν το θεώρημα κωδικοποίησης πηγής του *Shannon*. Αυτές που το τηρούν εξασφαλίζουν κωδικοποίηση χωρίς απώλεια πληροφορίας (*lossles*), ενώ οι *lossy* τεχνικές ή αλλιώς τεχνικές κβαντισμού, προσπαθούν να συμπιέσουν ακόμα περισσότερο τα σύμβολα εξόδου της πηγής, με τίμημα την απώλεια κάποιου ποσού πληροφορίας.

- Πιο γνωστές τεχνικές κβαντισμού είναι ο βαθμωτός (ομοιόμορφος και μη) και ο διανυσματικός ο οποίος υπερέχει του πρώτου. Τεχνικές κωδικοποίησης φωνής κωδικοποιούν απ'ευθείας τη κυματομορφή ή παράγουν τις παραμέτρους του φωνητικού σωλήνα, του μοντέλου της φωνής.

- Σε αντίθεση με τις συνήθεις τεχνικές κωδικοποίησης, που σαν σκοπό έχουν την πιστή αναπαραγωγή του σήματος της φωνής, η κωδικοποίηση που εφαρμόζουμε υπηρετεί ένα διαφορετικό σκοπό : Την μέγιστη συμπίεση με σκοπό την καλύτερη δυνατή αναγνώριση .

- Το σχήμα κωδικοποίησης που χρησιμοποιήσαμε, πηγαίνοντας ένα βήμα παραπέρα, κβαντίζει τα ακουστικά διανύσματα του *front-end*, προκειμένου να μειωθεί ακόμα περισσότερο το *bandwidth* που απαιτείται για την μετάδοσή τους.

- Για τη συμπίεση, χρησιμοποιείται σύστημα διανυσματικού κβαντισμού βασισμένο στην χρήση υποδιανυσμάτων (*product VQ* τεχνική). Στην διάρκεια της εκπαίδευσης κατασκευάζονται πρότυπα διανύσματα με τη βοήθεια του *k-means* αλγορίθμου. Κατά τη διαδικασία του κβαντισμού τα υποδιανύσματα κβαντίζονται στο καταλληλότερο πρότυπο υποδιάνυσμα. Στο Κεφάλαιο 3 θα δούμε πως υλοποιήθηκε η κωδικοποίηση με το

σύστημα *Yarrow* καθώς και τη χρήση του για την εκπαίδευση και τον κβαντισμό.

- Η αξιολόγηση της κωδικοποίησης γίνεται με την αναγνώριση του κωδικοποιημένου σήματος. Τα πειράματα για την αξιολόγηση της κωδικοποίησης αναλύονται στο Κεφάλαιο 4.

2.12 Παραπομπές

Δυο κλασικές εργασίες στη χρήση διανυσματικού κβαντισμού παρουσιάζονται στα [6] και [7]. Στη πρώτη δίνεται έμφαση στις διαφορές βαθμωτού και διανυσματικού κβαντισμού και παρουσιάζεται η αποτελεσματικότητα του δεύτερου, καθώς βελτιστοποιεί την παρουσία γραμμικής και μη γραμμικής εξάρτησης, σχήματος συνάρτησης πυκνότητας πιθανότητας (*probability density function*) και της διάστασης του διανύσματος. Στη δεύτερη παρουσιάζεται μια σειρά διαφορετικών τεχνικών βασισμένων σε διανυσματικό κβαντισμό. Για πιο ολοκληρωμένη εικόνα το [8] είναι ένα βιβλίο αφιερωμένο σε τέτοιες τεχνικές. Τεχνικές κωδικοποίησης φωνής παρουσιάζονται επίσης στο βιβλίο [9] , ενώ στο [5] δίνεται έμφαση στις τεχνικές κωδικοποίησης πηγής.

Κεφάλαιο 3

Υλοποίηση της κωδικοποίησης - το σύστημα YARROW

3.1 Εισαγωγή

Η υλοποίηση της κωδικοποίησης που παρουσιάσαμε προηγούμενα έγινε επεκτείνοντας το σύστημα φωνής *Yarrow*. Το *Yarrow* σύστημα άρχισε να αναπτύσσεται στο *S.R.I* από το *Leonardo Neumeyer* το 1997. Σκοπός του *project* είναι να δημιουργηθεί ένα πλαίσιο από γενικές βιβλιοθήκες (*Application Programming Interface - API*) σε τομείς όπως ψηφιακή επεξεργασία σήματος, αναγνώριση προτύπων & στατιστική μοντελοποίηση, για χρήση τόσο σε έρευνα όσο και σε γενικότερες εφαρμογές. Η δημιουργία του βασίστηκε στην ανάγκη για την ύπαρξη ενός *front-end* μηχανισμού επεξεργασίας φωνής που να είναι εύκολα μεταφέρσιμος σε διάφορες πλατφόρμες, εύκολος στη χρήση και να έχει μικρές απαιτήσεις σε πόρους κάτι που οδήγησε στην επιλογή της γλώσσα προγραμματισμού *Java* (*www.java.sun.com*) για την υλοποίησή του.

Σε αυτή την ενότητα περιγράφεται η βασική δομή του συστήματος και αναλύεται ο τρόπος επέκτασης του, με σκοπό τη δημιουργία μιας πλατφόρμας που να επιτρέπει την ερευνητική διαδικασία πάνω στο θέμα του διανυσματικού κβαντισμού των παραμέτρων της φωνής. Αρχικά περιγράφεται εκτενώς η αρχιτεκτονική του κώδικα που αναπτύχθηκε για την υποστήριξη διανυσματικού κβαντισμού και παρουσιάζεται η διασύνδεση με το σύστημα αναγνώρισης φωνής *Decipher*, το οποίο προσαρμόστηκε κατάλληλα για αυτό το σκοπό. Επίσης αναλύεται η χρήση του νέου συστήματος *Yarrow* για την διεξαγωγή πειραμάτων κωδικοποίησης φωνής. Στόχος είναι να χρησιμοποιηθεί το *Yarrow* στον πελάτη και το *Decipher* στον εξυπηρετητή σύμφωνα με το σενάριο πελάτη-εξυπηρετητή που αναφέρθηκε στο εισαγωγικό κεφάλαιο.

3.2 Βιβλιοθήκες του αρχικού συστήματος

Οι βιβλιοθήκες (*packages*) του αρχικού συστήματος *Yarrow* είναι οι εξής:

- **sri.star.dsp**
  Αυτή είναι η βασική βιβλιοθήκη του *front-end*. Ο σχεδιασμός βασίζεται σε αυτή του *Decipher*. Το *front-end* χωρίζει τη κυματομορφή σε σταθερού μήκους τμήματα (*frames*), και υπολογίζει διάφορες αναπαραστάσεις της αρχικής κυματομορφής που ονομάζονται *features*. Το *front-end* υλοποιείται σαν μια συλλογή από τέτοια *feature objects*. Η συλλογή αυτή αποτελείται από ένα γράφο με καθορισμένη τοπολογία. Στο γράφο αυτό κάθε *feature* είναι ένας κόμβος (*node*) με ένα σύνολο από *arcs* σαν είσοδο, από άλλα *features*. Για κάθε *frame*, προκειμένου να υπολογιστεί κάποιο *feature*, πρέπει να υπολογιστούν όλα τα άλλα από τα οποία εξαρτάται σύμφωνα με το γράφο. Η χρήση σχήματος κρυφής μνήμης επιταχύνει τη παραπάνω διαδικασία.

  Η παραγωγή *features* βασίζεται στη χρήση μιας σειράς φίλτρων (*bank of filters front-end processor*) η οποία είναι παρούσα στις περισσότερες υλοποιήσεις συστημάτων φωνής. Σκοπός είναι η εκτίμηση της ενέργειας του σήματος φωνής στις διάφορες μπάντες συχνότητας (συνήθως *Mel frequency*). Έτσι τελικά σχηματίζεται το *MFCC* διάνυσμα που είδαμε σε προηγούμενο κεφάλαιο. Για υψηλής ποιότητας φωνή, χρησιμοποιούνται 25 τέτοια φίλτρα. Κάθε ένας από τους 12 *cepstral* συντελεστές προκύπτει από το γινόμενο ενός διανύσματος βαρών για κάθε από τις 25 ενέργειες με το διάνυσμα των 25 ενεργειών. Στο διάνυσμα των 12 συντελεστών προστίθεται και το 'άθροισμά' τους, αποκτώντας έτσι το τελικό διάνυσμα με τους 13 συντελεστές (*cepstrum feature*). Υπάρχουν και άλλες μορφές *cepstrum* όπως το *rasta* και *zero mean* το οποίο και χρησιμοποιήσαμε. Από εδώ και πέρα λέγοντας *cepstrum* εννοούμε το τελευταίο. Το άλλο *MFCC* διάνυσμα που χρησιμοποιούμε είναι το *Single* και αποτελείται από το *cepstrum* μαζί με τις πρώτες και δεύτερες παραγώγους (σύνολο 39 συντελεστές).

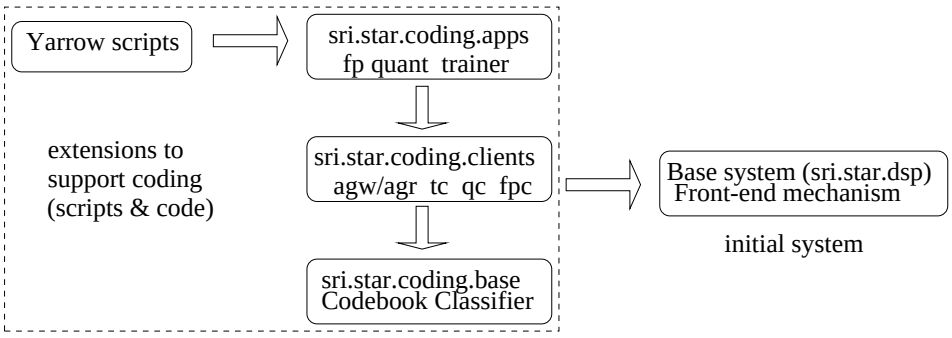
- **sri.star.util**
  Αυτή η βιβλιοθήκη περιέχει αρκετές βοηθητικές κλάσεις όπως στατιστικές μεθόδους για διανύσματα & πίνακες, και κλάσεις για διάβασμα - γράψιμο κυματομορφών, με *NIST Sphere* τύπο αρχείων.

- **sri.star.wavsource**
  Γενική βιβλιοθήκη με σκοπό την υποστήριξη περισσότερων ειδών εισόδου εκτός από κυματομορφές *NIST*.

3.3 Επέκταση του συστήματος για υποστήριξη VQ

3.3.1 Σκοπός

Σκοπός της επέκτασης του συστήματος *Yarrow* είναι να μπορεί να εφαρμοστεί διανυσματικός κβαντισμός στα παραγόμενα από το *front-end features* (κάτι το οποίο παρέχεται ήδη από την αρχική μορφή του *Yarrow*), ώστε να είναι δυνατή η μέγιστη συμπίεση του σήματος της φωνής, εξασφαλίζοντας παράλληλα υψηλή ακουστική ανάλυση. Για την διεξαγωγή των πειραμάτων, θα πρέπει να είναι δυνατή η επιλογή διαφόρων σχημάτων κωδικοποίησης. Από τη μεριά του *Yarrow* θα πρέπει να παρέχεται η δυνατότητα εκπαίδευσης των *codebooks* για το *training set*, καθώς και η χρήση τους για την κβαντοποίηση του *test set*. Από την άλλη πλευρά, τα κβαντισμένα αρχεία θα δίνονται σαν είσοδος στον αναγνωριστή του *Decipher*, για την αξιολόγηση του σχήματος κωδικοποίησης με το οποίο γίνονται κάθε φορά τα πειράματα. Ο τρόπος επέκτασης του *Yarrow* για την υλοποίηση του σχήματος κωδικοποίησης αυτής της εργασίας φαίνεται στο Σχήμα 3.1, όπου παρουσιάζεται η αρχιτεκτονική του κώδικα, ο τρόπος χρήσης του από τα πειράματα μέσω των *scripts* που αναπτύχθηκαν καθώς και η διασύνδεση με το αρχικό σύστημα *Yarrow*.



Σχήμα 3.1: **Αρχιτεκτονική Yarrow συστήματος** Η επέκταση αφορά στην προσθήκη των *sri.star.coding packages* και των τελικών *scripts*. Όπως θα δούμε στη συνέχεια το *fpc package* αποτελεί το συνδετικό κρίκο ανάμεσα στα 2 συστήματα.

3.4 Αρχιτεκτονική συστήματος και υλοποίηση

Η πλατφόρμα του *Yarrow* χρησιμοποιήθηκε για την υλοποίηση του σχήματος κωδικοποίησης. Οι 3 παραπάνω βιβλιοθήκες αποτέλεσαν μια πραγματική βάση

για την περαιτέρω επέκταση. Οι νέες βιβλιοθήκες που προστέθηκαν έχουν επεκτείνει το όνομα *sri.star* με το *sri.star.coding* για να είναι φανερό ότι ανήκουν στην βιβλιοθήκη κωδικοποίησης. Η βιβλιοθήκη αποτελείται από τις βασικές (*core*) κλάσεις (*sri.star.coding.base*), τις βοηθητικές (*sri.star.coding.utils*), τους πελάτες (*sri.star.coding.clients.**) & τις εφαρμογές (*sri.star.coding.apps.**). Αναλυτικότερα τα *packages* που αναπτύχθηκαν είναι τα ακόλουθα :

- *sri.star.coding.base*
- *sri.star.coding.clients.agw*
- *sri.star.coding.clients.agr*
- *sri.star.coding.clients.fpc*
- *sri.star.coding.clients.qc*
- *sri.star.coding.clients.net*
- *sri.star.coding.clients.tc*
- *sri.star.coding.clients.utils*
- *sri.star.coding.utils*
- *sri.star.coding.apps.fp*
- *sri.star.coding.apps.quant*
- *sri.star.coding.apps.trainer*
- *sri.star.coding.apps.utils*

3.5 Βιβλιοθήκη sri.star.coding.base

Αποτελεί πράγματι τη βάση του συστήματος. Περιέχει τις κλάσεις που αποτελούν το συνδετικό κρίκο ανάμεσα στην εκπαίδευση και τον κβαντισμό. Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, ο συνδετικός αυτός κρίκος δεν είναι άλλος από το *codebook*. Έτσι, αυτή η βιβλιοθήκη ορίζει και εκπαιδεύει τα *codebooks*. Οι βασικές κλάσεις είναι :

TrainerInformation
: Κλάση υπεύθυνη για τη συλλογή των στατιστικών ιδιοτήτων του υποδιανύσματος στο *training set*, απαραίτητες για την εκπαίδευση των *codebooks*

Codebook

Το *codebook* δεν είναι παρά μια συλλογή από 2^B διανύσματα μήκους L (*centroids*), αυτά που αντιπροσωπεύουν καλύτερα με B *bits* πληροφορίας το υποδιάνυσμα στο *train set*. Έτσι αυτή η κλάση εκτός από τον ορισμό, υλοποιεί μεθόδους όπως :

- τον διαμοιρασμό (*split*) ενός *centroid* σε 2 νέα (που χρησιμοποιείται κατά την εκπαίδευση),
- τον κβαντισμό ενός υποδιανύσματος εισόδου στο πλησιέστερο από τα *centroids* του *codebook*,
- το διάβασμα/γράψιμο του *codebook*

Classifier

Είναι το αντικείμενο που υλοποιεί τον *k-means* αλγόριθμο για την εκπαίδευση του *codebook*.

3.6 Βιβλιοθήκη πελατών (sri.star.coding.clients.*)

Οι πελάτες είναι κλάσεις που βρίσκονται ένα επίπεδο πάνω από τις βασικές κλάσεις και ένα επίπεδο κάτω από τις εφαρμογές στο *abstraction layer* του *coding package* (βλέπε Σχήμα 3.1). Με λίγα λόγια υλοποιούν όλες τις εξειδικευμένες λειτουργίες που χρειάζονται οι εφαρμογές.

3.6.1 Πελάτες αρχείων

Το *Yarrow* σύστημα, στη βασική του μορφή δεν παράγει αρχεία, απλώς κάνει *front-end* επεξεργασία σε αρχεία κυματομορφών. Παρ' όλα αυτά, καθορίζει κάποιο τύπο αρχείων που όμως δεν το υλοποιεί. Το φορματ καθορίζει *ascii* αρχεία, τα οποία είναι σε συμπιεσμένη μορφή. Οι δυο επόμενες κατηγορίες υλοποιούν το διάβασμα & γράψιμο *features* από/σε αυτό το φορματ. Λέγοντας *feature*, εννοούμε κβαντισμένα ή όχι *features*, όπως π.χ. *cepstrum, centroid coded* ή *index coded cepstrum*. Έτσι υπάρχει ένα μοναδικό καλά ορισμένο και απλό σχετικά φορματ, για όλα τα *features* του *Yarrow* συστήματος.

Το φορματ καθορίζει κάποιο *header* και τα δεδομένα. Το *header* δίνει πληροφορία για τον τύπο του *feature* (*integer/float*), το όνομα (*cepstrum/Single*), το μέγεθος, και τον αριθμό των *frames*. Στο πρώτο παράδειγμα, έχουμε 175 *frames* από *cepstrum*. Το *Compute Deltas* δηλώνει ότι πρέπει να υπολογισθούν οι πρώτες και δεύτερες παράγωγοι, πριν γίνει η αναγνώριση. Παράδειγμα αρχείου με *cepstrum* φαίνεται στη συνέχεια :

VectorList
Type float
Name cepstrum
VectorSize 13
NumVectors 175
ComputeDeltas true
EndHeader
-600.9399 1.2876966 0.5145594 -2.0349026 22.101751 -5.7347784 -0.022707298
15.5569315 -26.187925 8.510943 1.3709744 -1.8681848 7.1525745
...
< τα υπόλοιπα *174 frames*>

Στο 2ο παράδειγμα που ακολουθεί, έχουμε 381 *frames* από *Single* που έχουν κβαντιστεί-διακριτοποιηθεί σε 9 υποδιανύσματα :

VectorList
Type int
Name Single
VectorSize 9
NumVectors 381
ComputeDeltas false
EndHeader
16 113 52 1 8 18 10 3 2
5 108 48 32 65 48 10 3 2
6 120 16 8 3 20 65 2 33
4 96 16 4 108 36 64 65 33
...
< τα υπόλοιπα *377 frames* >

Τα αρχεία αυτά έχουν ονόματα με τις εξής καταλήξεις :

cepstrum	.cep
Single	.Single
Centroid Coded Cepstrum	.ccc
Centroid Coded Single	.ccs
Index Coded Cepstrum	.icc
Index Coded Single	.ics

Πίνακας 3.1: Παραγώμενα features και καταλήξεις αντίστοιχων αρχείων

Οι 2 βιβλιοθήκες που σχετίζονται με το διάβασμα και γράψιμο των αρχείων είναι οι :

- **sri.star.coding.clients.agw** *Ascii Gziped Writer* κλάσεις. Υλοποιούν το γράψιμο των παραγόμενων αρχείων στο παραπάνω φορματ.
- **sri.star.coding.clients.agr** *Ascii Gziped Reader* κλάσεις. Υλοποιούν το διάβασμα από αρχεία στο παραπάνω φορματ.

3.6.2 Πελάτες front-end

Πρόκειται για το *package* **sri.star.coding.clients.fpc**.
feature Produce Client (fpc) κλάσεις, όπως :
CepstrumProducerClient,
SingleProducerClient.
Με άλλα λόγια κλάσεις που παράγουν τα μη-κβαντισμένα *features* όπως *cepstrum* και *Single*. Οι κλάσεις αυτές είναι οι μόνες που χρησιμοποιούν άμεσα το *front-end* και αποτελούν τον ενδιάμεσο κρίκο ανάμεσα στο *front-end* & το *coding* υποσύστημα. Έτσι απομονώνεται το ένα υποσύστημα από το άλλο. Με αυτό το τρόπο όλες οι υπόλοιπες κλάσεις του *coding package*, δεν χρειάζονται να ξέρουν απολύτως τίποτα για το *front-end*.

3.6.3 Πελάτες κβαντισμού

Πρόκειται για το *package* **sri.star.coding.clients.qc**.
Quantizer Client (qc) κλάσεις, όπως :

QuantizerClient,
Είναι μια *abstract* κλάση που επεκτείνεται από τις υπόλοιπες του *package* και υλοποιεί τα ακόλουθα μεταξύ άλλων :

- το διάβασμα του *configuration*, που καθορίζει πόσα και ποια *codebooks* θα χρησιμοποιηθούν για τον κβαντισμό (σχήμα κβαντισμού, βλέπε παράγραφο 2.9.1).
- το φόρτωμα των *codebooks*, των προτάσεων του test-set και ότι άλλη αρχικοποίηση, είναι απαραίτητη.

CentroidQuantizerClient, IndexQuantizerClient.
Οι δυο αυτές κλάσεις υλοποιούν τα δυο είδη κβαντισμού, δηλαδή την απεικόνιση του κάθε υποδιανύσματος στο πλησιέστερο *centroid* του *codebook*, με έξοδο είτε το ίδιο το *centroid (centroid coded)*, είτε στο δείκτη του *centroid (index coded)*.

3.6.4 Πελάτες εκπαίδευσης

Πρόκειται για το **sri.star.coding.clients.tc**.
Κλάσεις όπως :

TrainerInfoClient,
: Ο *TrainerInfoClient* συλλέγει τις στατιστικές ιδιότητες του υποδιανύσματος στο *train set*. Οι πληροφορίες αυτές είναι απαραίτητες για την εκπαίδευση.

TrainerClient,
: Οι λειτουργίες του *TrainerClient* είναι :

- αρχικοποίηση πελάτη,
- καθορισμός σχήματος εκπαίδευσης (πόσα και ποια υποδιάνυσμα, μέγεθος *codebooks*, βλέπε 2.9.1)
- διάβασμα στατιστικών ιδιοτήτων του υποδιανύσματος,
- διάβασμα των δεδομένων εκπαίδευσης,
- δημιουργία των *centroids (training)*,
- γράψιμο παραγόμενων *codebooks*, κ.α.

IndexTrainerClient.
: Λειτουργεί σαν *wrapper* του *TrainerClient* επιτρέποντας την επιλογή των διανυσμάτων που θέλουμε να εκπαιδεύσουμε

Τέλος,υπάρχουν αρκετές κλάσεις που επεκτείνουν τον *TrainerClient*, αυξάνοντας την λειτουργικότητά του.

3.6.5 Υπόλοιποι πελάτες

Για τις υπόλοιπες λειτουργίες υπάρχουν πελάτες όπως αυτοί που ανήκουν στα πακέτα :

- **sri.star.coding.utils**
  πελάτες που περιέχουν πιο γενικές βοηθητικές λειτουργίες.
- **sri.star.coding.net**
  πελάτες υπεύθυνοι για την επικοινωνία *applet*-αναγνωριστή σε δίκτυο.
- **sri.star.coding.plugin**
  πελάτες που χρησιμεύουν στη λειτουργία του *applet*.

3.7 Βιβλιοθήκη εφαρμογών (sri.star.coding.apps.*)

Είδαμε παραπάνω ότι υπάρχουν πολλά επίπεδα στα οποία είναι οργανωμένος ο κώδικας. Όπως οι πελάτες βρίσκονται ένα επίπεδο πάνω από το *base package*, παρόμοια το *apps package* βρίσκεται ένα επίπεδο πάνω από τους πελάτες και οργανώνονται σε αντίστοιχα *packages* όπως εφαρμογές *front-end*, κβαντισμού, εκπαίδευσης (βλέπε σχήμα 3.1).

Κρύβοντας όλη την δύσκολη υλοποίηση στους πελάτες, οι εφαρμογές απλά συνδυάζουν ένα ή περισσότερους πελάτες για να υλοποιήσουν μια σχετικά πολύπλοκη διαδικασία, απλά με μερικές γραμμές κώδικα !

- **sri.star.coding.apps.fp**
  Εφαρμογές που χρησιμοποιούν τα
  sri.star.coding.clients.fpc &
  sri.star.coding.clients.agw packages, δηλαδή εφαρμογές που παράγουν & γράφουν μη-κβαντισμένα *features* όπως *Single* & *cepstrum*. Κλάσεις όπως *SingleProducer* & *CepstrumProducer*.

- **sri.star.coding.apps.qc**
  Εφαρμογές που υλοποιούν κβαντισμό. Υπάρχουν όλοι οι δυνατοί συνδυασμοί, ανάλογα με την είσοδο (κυματομορφή ή μη-κβαντισμένα *feature*) & το είδος της κβαντοποίησης (δείκτης ή διάνυσμα). π.χ. :

  WaveformToCentroidCodedSingleQuantizer :
  Διαβάζει από κυματομορφή, παράγει *Single feature* και μετά το κβαντίζει σε *cenroid*. Χρησιμοποιεί πελάτες όπως :
  SingleProducerClient
  CentroidQuantizerClient
  SingleWriterClient

  CepstrumToIndexCodedCepstrumQuantizer :
  Διαβάζει από *cepstrum* αρχείο τα *frames* και τα κβαντίζει σε δείκτες. Χρησιμοποιεί πελάτες όπως :
  CepstrumReaderClient
  IndexQuantizerClient
  IndexCodedCepstrumWriterClient

- **sri.star.coding.apps.utils**
  Χρήσιμες κλάσεις όπως : *CepstrumToSingle*(μετατροπέας), *CodebookInfo, ShowTrainerInfo*

- **sri.star.coding.apps.trainer**
  Κλάσεις όπως *Trainer, InitTrainer*, κ.α. Βασίζονται στους αντίστοιχους πελάτες *sri.star.coding.clients.tc*. Για παράδειγμα, ο *Trainer* το μόνο που κάνει, είναι να δημιουργεί ένα αντικείμενο τύπου *IndexTrainerClient* και να καλεί την μέθοδο *produceCodebooks()* (όπως φαίνεται και στο παράδειγμα που ακολουθεί).

```
package sri.star.coding.apps.trainer;

import sri.star.coding.clients.trainer.*;
import sri.star.util.*;

/**
 * Produces the desired codebooks for all subVectors
 *
 * @see sri.star.coding.clients.trainer.IndexTrainerClient
 * @author  Perakakis Manolis
 * @version coding (1.0)
 */
 public class Trainer {

 public static void main(String args[]) {
    String config
    String tmp "";
    int index;

    if(args.length == 2) {
      config = args[0];
      tmp = args[1];
    }
    else
      ErrorHandler.fatalError(null,
        "USAGE: java Trainer <config_file> <index> ");
    try {
      index = (new Integer(tmp)).intValue();
      IndexTrainerClient itc = new IndexTrainerClient(config,
index);
      itc.produceCodebooks();
    }catch (Exception e) {
      e.printStackTrace();}
```

```
  }
}
```

3.8 Πειράματα VQ με το Yarrow

Μέχρι τώρα εξετάστηκε το *Yarrow* σε επίπεδο κώδικα. Όμως για να τρέξουν τα πειράματα χρειάζεται συχνά αρκετή προετοιμασία. Έτσι, πέρα από το *API*, γράφτηκαν τα απαραίτητα *scripts* τα οποία βρίσκονται ένα επίπεδο πάνω από τις εφαρμογές και απευθύνονται στον χρήστη, σε αντίθεση με το *API* που απευθύνεται στον *developer*. Τα πειράματα που θέλει να τρέξει κάποιος για να ελέγξει την επίδοση της κωδικοποίησης για κάποιο σχήμα, βασίζονται σε μια αλληλουχία ενεργειών που περιγράφεται στις 5 επόμενες παραγράφους :

3.8.1 Παραγωγή επιθυμητού feature από αρχεία κυματομορφών

Τόσο για το *train* όσο & για το *test set*. Αυτό γίνεται προκειμένου να αποφεύγεται η χρήση του *front-end* κάθε φορά που θέλουμε να δοκιμάσουμε ένα νέο σχήμα κωδικοποίησης[1] θα χρησιμοποιήσουμε για το κάθε διάνυσμα). Αντίθετα έχοντας παράγει το επιθυμητό *feature* (π.χ. *Single*) τόσο για το *test set* όσο και για το *train set* , το μόνο που χρειάζεται να γίνεται κάθε φορά είναι μόνο η λειτουργία του κβαντισμού/εκπαίδευσης και όχι επιπλέον και η λειτουργία της παραγωγής του *feature* μέσω του *front-end* μια διαδικασία σχετικά χρονοβόρα. Με αυτό το τρόπο σε λιγότερο χρόνο μπορούμε να κάνουμε περισσότερα πειράματα. Το επιθυμητό *feature* μπορεί να είναι *cepstrum* ή *Single*. Το *script* που κάνει τα παραπάνω είναι το **go.produce.Single** για το *Single feature* και το **go.produce.cepstrum** για το *cepstrum*. Απαιτεί **configuration** αρχείο της μορφής :

input_files in.wavs
output_files out.cepstrum

δηλαδή απλά περιέχει τις λίστες με τα αρχεία κυματομορφών και τα παραγώμενα αρχεία.

[1] Εδώ καλό είναι να θυμίσουμε τους όρους σχήμα κωδικοποίησης, εκπαίδευσης και κβαντισμού. Ο 1ος είναι πιο ευρύς και περιλαμβάνει τον συνδυασμό των επόμενων. Το σχήμα εκπαίδευσης καθορίζει τις επιλογές για την εκπαίδευση των *codebooks* (πώς θα ορίσουμε τα υποδιανύσματα, το μέγεθος των *codebooks*), ενώ το σχήμα κβαντισμού καθορίζει ποιά από τα διαθέσιμα για κάθε υποδιάνυσμα *codebooks* θα χρησιμοποιήσουμε στο κβαντισμό.

3.8.2 Προετοιμασία εκπαίδευσης

Και αυτό το βήμα αρκεί να γίνει μια φορά. Εδώ χρειάζεται να παραχθούν οι στατιστικές ιδιότητες του *feature* στο *train set*. Αφού τρέξει το *script* θα παραχθεί το καθορισμένο αρχείο στατιστικών ιδιοτήτων το οποίο θα διαβαστεί αργότερα από τον εκπαιδευτή. Το *script* για *cepstrum feature* είναι το **go.InitCepstrumTrainer**, ενώ για το *Single feature* είναι το **go.InitSingle Trainer**.

3.8.3 Διαδικασία εκπαίδευσης

Για κάθε σχήμα εκπαίδευσης που θέλουμε να δοκιμάσουμε, πρέπει να τρέξουμε τόσους *trainers*, όσος και ο αριθμός των υποδιανυσμάτων, παράγοντας τα αντίστοιχα *codebooks*.

Αφού καθοριστούν τα υποδιανύσματα, θα πρέπει να καθοριστεί και το μέγεθος του *codebook*, δηλαδή ο αριθμός των *centroids* που περιέχει το *codebook*. Αν για παράδειγμα χρειάζονται 64 *centroids*, θα γραφτούν για αυτό το υποδιάνυσμα τα *codebooks* με 1 έως 6 *bits*, έτσι ώστε αργότερα στο κβαντισμό να έχουμε διαθέσιμα όλα τα δυνατά *codebooks* και να μη ξαναχρειαστεί να παράγουμε το *codebook* με π.χ. 16 *centroids* αν αυτά χρειαστούν.

Η ονοματολογία των *codebooks* που γράφονται είναι η εξής :
$< codebook.id.numberOfBits >$, όπου *id* είναι το *id* του υποδιανύσματος όπως καθορίζεται στο *Coding* αρχείο. Έτσι για παράδειγμα, εάν πρόκειται να παραχθούν *codebooks* με 6 *bits* για το 3ο υποδιάνυσμα, τα αρχεία που θα παραχθούν θα είναι τα:

codebook.3.1
codebook.3.2
codebook.3.3
codebook.3.4
codebook.3.5
codebook.3.6

Το **Coding** αρχείο είναι αυτό που καθορίζει το σχήμα εκπαίδευσης και έχει την εξής μορφή :

2 1 2 3
2 3 4 3
3 5 6 7 3
3 8 9 10 3
3 11 12 13 3

το φορματ είναι : <μέγεθος υποδιανύσματος> < στοιχεία του *feature* διανύσματος που το αποτελούν> <αριθμός *bits* > . Η γραμμή καθορίζει και το *id* του υποδιανύσματος. Έτσι π.χ. η γραμμή < 3 8 9 10 3 > δηλώνει ότι το υποδιάνυσμα 4 αποτελείται από τα εξής 3 στοιχεία του *feature* διανύσματος : 8 9 10, και το *codebook* που θα παραχθεί θα είναι 3 *bits*, δηλαδή θα περιέχει 8 *centroids*.

Εκτός από το παραπάνω αρχείο απαιτείται και κάποιο *configuration* αρχείο της μορφής :

feature_type	cepstrum
input_files	trainerFiles
trainer_info_file	../../InitTrainer/ti.cepstrum
coding_configuration	Coding
codebook_path	codebooks/cepstrum/
e	0.01
repetitions	4
total_number_of_codebooks	5

Το *feature_type* καθορίζει τι είδους *feature* έχουμε(*Single,cepstrum*).
Το *input_files* είναι η λίστα με τα αρχεία του *train set*.
Το *trainer_info_file* είναι το αρχείο με τις στατιστικές ιδιότητες που αναφέρθηκε παραπάνω.
Το *coding_configuration* είναι το παραπάνω *Coding* αρχείο.
Το *codebook_path* καθορίζει που θα γραφτούν τα *codebooks*.
Το *e* είναι μια σταθερά που καθορίζει πόσο θα διαφέρουν τα 2 νέα *centroids* μετά το *splitting* του αρχικού.
Το *repetitions* είναι ο αριθμός επαναλήψεων στον αλγόριθμο εκπαίδευσης.
Το *total_number_of_codebooks* είναι ο συνολικός αριθμός των υποδιανυσμάτων.

Τα *scripts* εκπαίδευσης είναι τα **go.SingleTrainer** για *Single features* και το **go.CepstrumTrainer** για *cepstrum features*.

3.8.4 Διαδικασία κβαντισμού

Αφού παράγουμε τα *codebooks* σύμφωνα με τη παραπάνω διαδικασία, είμαστε έτοιμοι να προχωρήσουμε στο επόμενο βήμα του κβαντισμού. Ο κβαντιστής θα πρέπει να φορτώσει τα κατάλληλα *codebooks*. Αυτό καθορίζεται σε *configuration* αρχείο (καθορίζει το σχήμα κβαντισμού), που έχει τη μορφή :

input_files	in.cepstrums
output_files	out.coded.cepstrums
codebook_path	../../trainer/codebooks/cepstrum/
number_of_codebooks	5
codebook1	3
codebook2	4
codebook3	5
codebook4	4
codebook5	4

Το *input_files* είναι η λίστα με τα αρχεία που θα κβαντιστούν (αυτά που φτιάξαμε στο 1ο βήμα)
Το *output_files* είναι η λίστα με τα ονόματα των κβαντισμένων αρχείων που θα παραχθούν
Το *codebook_path* είναι η τοποθεσία απ'οπου θα φορτωθούν τα *codebooks*

Οι επόμενες 5 γραμμές καθορίζουν τα *codebooks* που θα χρησιμοποιηθούν. Αυτά είναι με βάση το φορματ που καθορίστηκε παραπάνω τα :

codebook.1.3
codebook.2.4
codebook.3.5
codebook.4.4
codebook.5.4

Το *script* του κβαντισμού είναι το **go.Quantizer**

3.8.5 Έλεγχος με Αναγνώριση

Τα αρχεία που παράγονται από τον κβαντισμό δίνονται σαν είσοδος στον αναγνωριστή για να ελέγξουμε την επίδοση του σχήματος κωδικοποίησης που χρησιμοποιήσαμε. Συνήθως φροντίζουμε να παράγουμε *codebooks* με περισσότερα *bits*, απ'οσα συνήθως πιστεύουμε ότι ίσως είναι αρκετά. Με αυτό το τρόπο εξασφαλίζουμε ότι η επίδοση της αναγνώρισης θα είναι σε υψηλά επίπεδα. Έχοντας εξασφαλίσει το παραπάνω, πρέπει να αφαιρέσουμε όση πλεονάζουσα πληροφορία υπάρχει, χωρίς όμως να επιτρέπουμε μείωση της επίδοσης του αναγνωριστή. Αυτός άλλωστε είναι και ο στόχος μας. Για να αφαιρέσουμε τη πλεονάζουσα πληροφορία επιλέγουμε *codebooks* με λιγότερα *bits*. Έτσι για κάθε σχήμα κωδικοποίησης (σχήμα εκπαίδευσης + σχήμα κβαντισμού), εκτελούμε τόσους κβαντισμούς και αναγνωρίσεις όσους χρειάζονται για να επιτύχουμε το στόχο που τέθηκε παραπάνω.

Κάποια σχήματα κωδικοποίησης δίνουν μη ικανοποιητικά αποτελέσματα. Αυτό που πρέπει να γίνει τότε, είναι να καθοριστεί ένα νέο σχήμα κωδικοποίησης και να επαναληφθεί η διαδικασία κβαντισμού και αναγνώρισης για αυτό το σχήμα. Αν χρειαστεί να γίνουν πειράματα σε άλλα *train/test sets*, θα πρέπει να επαναληφθούν τα 2 πρώτα βήματα.

3.9 Αναγνώριση με το Decipher

Το σύστημα αναγνώρισης που χρησιμοποιήθηκε είναι αυτό του *Decipher*. Προκειμένου να γίνουν τα πειράματα έγιναν αλλαγές για να υποστηρίζουν τα νέα κβαντισμένα *features* που δημιουργήθηκαν με το *Yarrow*. Επίσης αφού όλα τα αρχεία που παράγονται με το *Yarrow* έχουν καθορισμένο φορματ, θα πρέπει το *Decipher* να μπορεί να τα διαβάσει και να τα αποκωδικοποιήσει. Επιπρόσθετα, στην περίπτωση που χρησιμοποιείται *index coded* κβαντισμός, θα πρέπει να γίνει διακριτοποίηση των μειγμάτων των κατανομών εξόδου των HMM μοντέλων. Σε αυτή τη περίπτωση, για κάθε σχήμα κωδικοποίησης χρειάζεται να ξαναγίνει η διακριτοποίηση των μειγμάτων έτσι ώστε να υπάρχει πάντα συνέπεια ανάμεσα στην είσοδο του συστήματος (*features*) και τα μοντέλα που χρησιμοποιούνται. Η διακριτοποίηση έγινε σε άλλη διπλωματική εργασία [18], ενώ στη παρούσα έγινε το απαραίτητο *interface* με το *Yarrow*. Για να υποστηρίζονται όλα τα παραπάνω, έγιναν οι απαραίτητες αλλαγές στην *front-end* βιβλιοθήκη του *Decipher*. Αυτές περιλαμβάνουν τα εξής :

- Επέκταση της *I/O* βιβλιοθήκης του *Decipher* για είσοδο από τα αρχεία που παράχθηκαν με το *Yarrow*.
- Δημιουργία ενός νέου *feature* για τη περίπτωση που έχουμε *index coded* κβαντισμό.

3.10 Συμπεράσματα

Σε αυτή την ενότητα παρουσιάστηκε η δουλειά που έγινε πάνω στο σύστημα *Yarrow*, τόσο σε επίπεδο κώδικα, όσο και πειραμάτων, καθώς και το *interface* με το σύστημα του *Decipher*. Δημιουργήθηκε τελικά μια πλατφόρμα που επιτρέπει την ερευνητική διαδικασία πάνω στο θέμα του Διανυσματικού Κβαντισμού των παραμέτρων της φωνής.

- Επεκτάθηκε το σύστημα ώστε να υποστηρίζει διανυσματικό κβαντισμό.
- Σε επίπεδο κώδικα, αναπτύχθηκε το *package sri.star.coding*, σαν μια κατάλληλα σχεδιασμένη ιεραρχία με πολλά επίπεδα το καθένα από τα

οποία επιτρέπει στον *developer* να κάνει αλλαγές χωρίς να επηρεάζεται η εικόνα του συστήματος από τον τελικό χρήστη. Το σύστημα είναι αρκετά ευέλικτο και επεκτάσιμο για μελλοντικές βελτιώσεις και προσθήκες.

- Σε επίπεδο πειραμάτων, σχεδιάστηκαν τα *configuration* αρχεία και τα *scripts* με τέτοιο τρόπο που να είναι εύκολο για τελικό χρήστη να τρέξει οποιοδήποτε πείραμα και να πειραματιστεί απλά αλλάζοντας κάποιες παραμέτρους.

- Έγιναν οι απαραίτητες αλλαγές στο *Decipher* σύστημα (επίσης τόσο σε επίπεδο κώδικα, όσο και πειραμάτων) για να επιτελεί αναγνώριση με τα παραγώμενα από το *Yarrow features*.

- Το *API* που αναπτύχθηκε, συνοδεύεται από τεκμηρίωση με πολλά σχόλια για κάθε κλάση χωριστά (σε μορφή *.html* σύμφωνα με τις συμβάσεις της *Java*), αλλά και μια σειρά από παραδείγματα πειραμάτων με επίσης όλα τα απαραίτητα σχόλια. Έτσι γίνεται εύκολο τόσο για τον *developer* να κατανοήσει και να επεκτείνει αν χρειαστεί τον κώδικα, αλλά και τον απλό χρήστη να τρέξει όποια πειράματα απαιτείται.

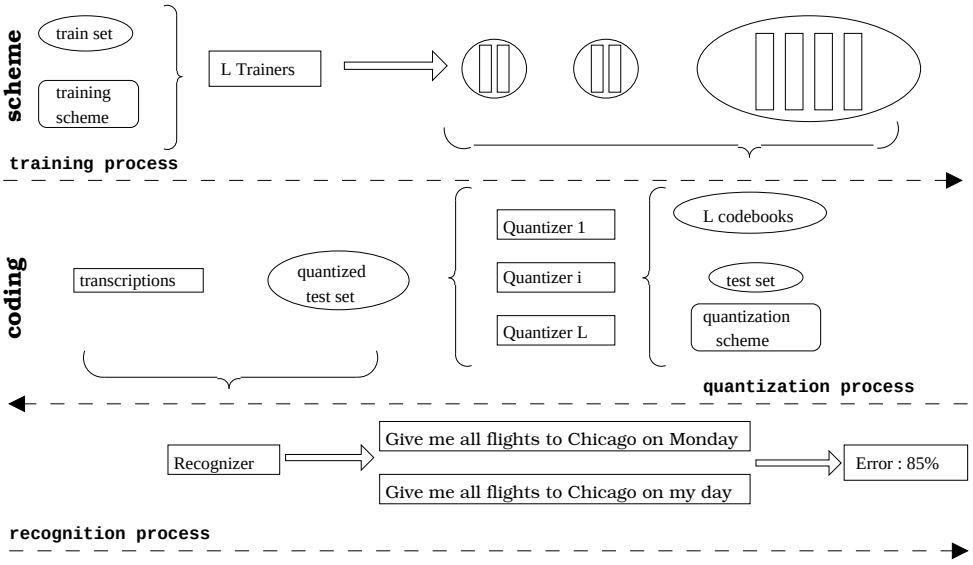
Κεφάλαιο 4

Πειράματα κωδικοποίησης

Στο Κεφάλαιο 2 αναλύσαμε το σχήμα κωδικοποίησης και τονίσαμε πως αποτελεί μέθοδο κωδικοποίησης φωνής προς αναγνώριση. Επισημάνθηκε επίσης ότι κάνουμε συνδυασμό τεχνικών κωδικοποίησης φωνής (επεξεργασία *front-end*) και τεχνικών κωδικοποίησης πηγής (διανυσματικός κβαντισμός) προκειμένου να μειώσουμε ακόμα περισσότερο τον ρυθμό μετάδοσης (ή ισοδύναμα να αυξήσουμε στο μέγιστο το βαθμό συμπίεσης), προσπαθώντας παράλληλα να κρατήσουμε υψηλά επίπεδα αναγνώρισης. Αναλύσαμε τεχνικές κβαντισμού με έμφαση στην τεχνική *product VQ* και είδαμε τις επιλογές που μπορεί να γίνουν κατά τον σχεδιασμό ενός συστήματος διανυσματικού κβαντισμού και τέλος παρουσιάσαμε τις διαδικασίες εκπαίδευσης και κβαντισμού για την κωδικοποίηση που προτείνουμε.

Στο Κεφάλαιο 3 μελετήσαμε την υλοποίηση της κωδικοποίησης που έγινε για το σύστημα *Yarrow* και είδαμε πως μπορούμε να το χρησιμοποιήσουμε για την εκπαίδευση και τον κβαντισμό. Είδαμε επίσης πως μπορούμε να το χρησιμοποιήσουμε με το σύστημα αναγνώρισης φωνής *Decipher* προκειμένου να αξιολογήσουμε ένα σχήμα κωδικοποίησης.

Σε αυτό το Κεφάλαιο θα δούμε αναλυτικά την διαδικασία των πειραμάτων και θα παρουσιάσουμε τα πειράματα που έγιναν, τα οποία αφορούν πειράματα τόσο βαθμωτού όσο και διανυσματικού κβαντισμού. Θα παραθέσουμε τον *bit allocation* αλγόριθμο που μας βοηθά στην εύρεση του επιτρεπόμενου ορίου συμπίεσης χωρίς απώλειες στην απόδοση αναγνώρισης και τέλος θα εξετάσουμε την επίδραση του θορύβου στην κωδικοποίηση. Αξίζει να σημειωθεί ότι στο Κεφάλαιο αυτό παρουσιάζονται τα 'τελικά' αποτελέσματα για τα οποία χρειάστηκε να γίνουν και πολλά άλλα πειράματα. Κάποια από αυτά που έγιναν με τη χρήση του *bit allocation* αλγορίθμου παρουσιάζονται στο Παράρτημα στο τέλος αυτής της εργασίας.



Σχήμα 4.1: **Διαδικασία πειραμάτων.** Οι *trainers* παράγουν τα L *codebooks*(όσα και τα υποδιανύσματα). Κάθε ένας από τους L κβαντιστές χρησιμοποιεί το αντίστοιχο *codebook* για να κβαντίσει το υποδιάνυσμα σε κάθε *frame* στο *test set*. Για κάθε κβαντισμένο αρχείο, ο αναγνωριστής συγκρίνει το *transcription*(περιγραφή πρότασης) με το αποτέλεσμα της αναγνώρισης και υπολογίζει το συνολικό σφάλμα.

4.1 Διαδικασία πειραμάτων

Στις παραγράφους 2.9.2 και 2.9.3 μιλήσαμε για την διαδικασία εκπαίδευσης και κβαντισμού. Σε αυτή τη παράγραφο θα δούμε την συνολική εικόνα της διαδικασίας πειραμάτων, η οποία εκτός από τις δυο προηγούμενες εμπεριέχει και τη διαδικασία αναγνώρισης, με την οποία και αξιολογούμε την αποτελεσματικότητα του σχήματος κωδικοποίησης που εξετάζουμε κάθε φορά. Η γενική διαδικασία των πειραμάτων φαίνεται στο Σχήμα 4.1 και περιλαμβάνει τις διαδικασίες εκπαίδευσης, κβαντισμού και αναγνώρισης σε κάθε ένα από τα 3 επίπεδα που την αποτελούν.

Για την εκπαίδευση και τον κβαντισμό χρησιμοποιήσαμε το *Yarrow*, ενώ για την αναγνώριση χρησιμοποιούμε το σύστημα αναγνώρισης φωνής *Decipher* (τόσο το *Yarrow* όσο και το *Decipher* περιγράφονται στο Κεφάλαιο 4). Για την απόδοση της κωδικοποίησης, χρησιμοποιούμε σαν μέτρο, το σφάλμα της αναγνώρισης (*Word Error Rate - WER*), το οποίο ορίζεται ως :

$WER = \frac{INS+DEL+SUB}{TOTAL} \times 100$. Είναι δηλαδή ο αριθμός των λάθος εισαγωγών, αντικαταστάσεων και παραλείψεων λέξεων, σε σχέση με το συνολικό αριθμό λέξεων στην πρόταση.

Τα πειράματα έγιναν στο *ATIS corpus*, το οποίο έχει ένα λεξικό 1500 λέξεων και είναι μέτριας δυσκολίας. Το *training set* του κβαντιστή αποτελείται από 700 προτάσεις και το *test set* από 400 προτάσεις. Στα πειράματα που παρουσιάζουμε γίνεται χρήση *Centroid Coded* κβαντισμού με τα συνεχή γκαουσιανά ακουστικά μοντέλα του *ATIS*. Όπως έχουμε τονίσει και προηγούμενα, αυτός ο τρόπος δεν προσφέρει πραγματική συμπίεση (αφού η έξοδος του κβαντισμού αναπαρίσταται με το διάνυσμα και όχι το δείκτη), αλλά είναι αναγκαία για τη διαδικασία των πειραμάτων και την εύρεση των πιο αποδοτικών σχημάτων κωδικοποίησης. Για τα πιο επιτυχημένα τέτοια σχήματα κωδικοποίησης, προχωρούμε στη διακριτοποίηση των ακουστικών μοντέλων (βλέπε παράγραφο 1.5.3), τα οποία προσφέρουν μεγάλη συμπίεση και γρηγορότερη αναγνώριση και θα χρησιμοποιούνται στη πράξη, σε εφαρμογές στηριζόμενες σε αυτή την εργασία.

Η γενική διαδικασία των πειραμάτων είναι η εξής :

- Στη διαδικασία εκπαίδευσης, χρησιμοποιούμε το *training set* και το δοσμένο σχήμα εκπαίδευσης(*training scheme*), με το οποίο καθορίζουμε πόσα και ποια υποδιανύσματα θέλουμε, καθώς και πόσα *bits* θα διαθέσουμε για την εκπαίδευσή τους για να παράγουμε τελικά τα *codebooks*.

- Στο σχήμα κβαντισμού, χρησιμοποιούμε το *test set* και το δοσμένο σχήμα κβαντισμού(*quantization scheme*), με το οποίο δηλώνουμε ποιά από αυτά τα *codebooks* θα χρησιμοποιήσουμε για να κάνουμε το κβαντισμό και ο κβαντιστής παράγει τα κβαντισμένα αρχεία που αντιστοιχούν σε αυτά του *test set*.

- Με τη διαδικασία αναγνώρισης, τα κβαντισμένα αυτά αρχεία περνάνε σαν είσοδος στον αναγνωριστή. Τα αποτελέσματα της αναγνώρισης συγκρίνονται με την περιγραφές των αρχείων και έτσι προκύπτει το ποσοστό λάθους της αναγνώρισης, το οποίο μας δείχνει κατά πόσο το σχήμα κωδικοποίησης(*coding scheme*) ήταν επιτυχές.

Στα πειράματα που ακολουθούν, αρχίζουμε με το *baseline* πείραμα, το οποίο χρησιμοποιεί το *test-set* (μη κβαντισμένα αρχεία) και το οποίο καθορίζει την ελάχιστη απόδοση αναγνώρισης που θα δεχόμαστε για τα πειράματα με κβαντισμό. Στα πειράματα αυτά, χρησιμοποιούμε τεχνικές βαθμωτού (ομοιόμορφου και μη) κβαντισμού, καθώς και τεχνικές *VQ* βασισμένες στη χρήση υποδιανυσμάτων (*productVQ*). Τέλος θα παρουσιάσουμε την χρήση του *bit-allocation* αλγορίθμου και πειράματα με θόρυβο.

4.2 Baseline πείραμα

Το *baseline* είναι το βασικό πείραμα που δείχνει την απόδοση για μη κβαντισμένα αρχεία. Σε όλα τα επόμενα πειράματα κβαντισμού, στόχος μας θα είναι να προσεγγίσουμε επίπεδα αναγνώρισης κοντά σε αυτά του *baseline* πειράματος, κάτι που θα αποδεικνύει ότι ο θόρυβος που εισαγάγει ο κβαντισμός είναι αρκετά μικρός ή δεν θα επηρεάζει αισθητά την διαδικασία της αναγνώρισης. Για τη διεξαγωγή του, χρησιμοποιήθηκαν αρχικά τηλεφωνικής ποιότητας (*toll quality*) δεδομένα, τόσο σε *PCM* κωδικοποίηση, όσο και σε *GSM*. Παρατηρούμε ότι αν και η διαφορά στο *bit-rate* είναι αρκετά μεγάλη, η διαφορά στην ακρίβεια αναγνώρισης είναι σχετικά μικρή. Το *baseline* μας ενδιαφέρει κυρίως για υψηλής ποιότητας (*high quality*) δεδομένα όπου η ακρίβεια αναγνώρισης ήταν **6.55**, για μεγαλύτερο όμως *bit-rate*.

Type of Encoding	Bit-rate (kbps)	Word-error Rate(%)
PCM toll quality(Mlaw)	64	12.7
GSM encoding	13	14.5
PCM high quality	256	6.55

Πίνακας 4.1: **Baseline πείραμα**. PCM(high & toll quality) & GSM encoding

4.3 Πειράματα βαθμωτού κβαντισμού

Αρχικά κβαντίσαμε τους συντελεστές τηλεφωνικής ποιότητας φωνής και εξετάσαμε την απόδοση για διαφορετικούς αριθμούς *bits/element*. Χρησιμοποιήσαμε τόσο ομοιόμορφο όσο και μη-ομοιόμορφο κβαντισμό. Στη περίπτωση του μη-ομοιόμορφου κβαντισμού, η απόσταση μεταξύ των σταθμών, προέκυψε από την κατανομή των συντελεστών του διανύσματος στο *train set*. Τα αποτελέσματα συνοψίζονται στο πίνακα 4.2.

Παρατηρούμε ότι για 5-8 *bits/element* η απόδοση της αναγνώρισης είναι αρκετά καλή, και στις 2 περιπτώσεις. Από εκεί και πέρα τα αποτελέσματα αλλάζουν αρκετά. Συγκεκριμένα, για τον ομοιόμορφο κβαντισμό βλέπουμε ότι μειώνοντας έστω και κατά 1 *bit/element*, η απόδοση πέφτει αμέσως στο *17.43* που ήδη είναι μη αποδεκτό, από άποψη αναγνώρισης. Αυτή η συμπεριφορά των αποτελεσμάτων σημαίνει ότι :
1. Κανένας συντελεστής δεν χρειάζεται περισσότερα από 5 *bits*.
2. Στον ομοιόμορφο κβαντισμό, χρησιμοποιώντας λιγότερα από 5 *bits/element*,

Bits/coef.	Bit-Rate (kbps)	Uniform (WER%)	Non-Uniform (WER%)
8	7.2	12.55	12.82
7	6.3	12.65	12.87
6	5.4	13.08	12.65
5	4.5	**13.14**	13.62
4	3.6	17.43	**13.19**
3	2.7	45.47	14.64
2	1.8	108.9	21.07

Πίνακας 4.2: **Bit-rates & WER για scalar quantization των συντελεστών του MFCC διανύσματος για τηλεφωνικής ποιότητας φωνή.** (Για τηλεφωνικής ποιότητας φωνή, το *MFCC* διάνυσμα έχει μέγεθος 9 : 8 *cepstral* συντελεστές και 1 συντελεστής για την ενέργεια)

Bits/coef.	Bit Rate (kbps)	Uniform (WER%)	Non-Uniform (WER%)
8	10.4	6.65	6.53
7	9.1	6.76	6.40
6	7.8	6.65	6.43
5	6.5	6.96	6.32
4	5.2	**6.96**	6.32
3	3.9	12.45	**6.88**
2	2.6	95.43	9.04

Πίνακας 4.3: **Bit rates & WER για scalar quantization των συντελεστών του MFCC διανύσματος για υψηλής ποιότητας φωνή.** (Για υψηλής ποιότητας φωνή, το *MFCC* διάνυσμα έχει μέγεθος 13 : 12 *cepstral* συντελεστές και 1 συντελεστής για την ενέργεια)

αφαιρούμε πληροφορία από ποιο σημαντικά *elements*, ενώ ίσως δίνουμε περισσότερα *bits* από ότι χρειάζεται σε άλλα *elements*. Αυτό έχει σαν αποτέλεσμα την ραγδαία αύξηση του *WER*.
3. Αντίθετα, στην περίπτωση του μη-ομοιόμορφου κβαντισμού, βλέπουμε ότι ακόμα και στα *2.7 Kbps* η απόδοση δεν πέφτει σημαντικά. Μάλιστα, είναι σχεδόν ίση με την απόδοση του *GSM* συστήματος το οποίο έχει ρυθμό μετάδοσης της τάξης των *13 Kbps*.

Στη συνέχεια χρησιμοποιήσαμε υψηλής ποιότητας φωνή και διεξάγαμε ανάλογα πειράματα (βλέπε Πίνακα 4.3). Παρατηρούμε πάλι, ότι στην περίπτωση του μη-ομοιόμορφου κβαντισμού έχουμε καλύτερα αποτελέσματα. Χρησιμο-

ποιώντας *5.2 & 3.9 kbps* αντίστοιχα για ομοιόμορφο και μη ομοιόμορφο κβαντισμό, υπερβαίνουμε κατά λίγο το *WER* στο *baseline* πείραμα (*6.55%*). Συμπερασματικά, βλέπουμε ότι χρησιμοποιώντας ένα σχετικά απλό σχήμα παίρνουμε πολύ καλά αποτελέσματα.

4.4 Πειράματα διανυσματικού κβαντισμού

Είδαμε στην παράγραφο 2.5.2, ότι η *product VQ* τεχνική, μπορεί να αποδειχθεί αρκετά αποτελεσματική, εφόσον επιλεγούν κατάλληλα οι παράμετροι αυτής της τεχνικής, που είναι ο καθορισμός των υποδιανυσμάτων και ο αριθμός *bits* που θα διαθέσουμε σε κάθε ένα από αυτά.

4.4.1 Επιλογή παραμέτρων συστήματος

Ο αριθμός των συνδυασμών που προκύπτει από την επιλογή των υποδιανυσμάτων στα οποία μπορούμε να χωρίσουμε το *MFCC* διάνυσμα και τον αριθμό των *centroids* ανά υποδιάνυσμα, είναι απαγορευτικά μεγάλος. Έτσι, έπρεπε να βρεθεί ένας συστηματικός τρόπος επιλογής των παραμέτρων.

4.4.2 Επιλογή υποδιανυσμάτων

Για την επιλογή των υποδιανυσμάτων υπήρξαν δυο προσεγγίσεις :

Η πρώτη προσέγγιση είναι να ομαδοποιήσουμε τα στοιχεία του *MFCC* διανύσματος με τέτοιο τρόπο, ώστε αυτά με τη μεγαλύτερη στατιστική συσχέτιση, να ομαδοποιηθούν στο ίδιο υποδιάνυσμα. Η προσέγγιση αυτή δεν έδωσε τα αναμενόμενα αποτελέσματα, καθώς αποδείχθηκε ότι οι συσχετίσεις μεταξύ των *elements* κυμαίνονταν σε πολύ μικρές τιμές. Αυτό επιβεβαιώνεται από τη χρήση *DCT* στο *front-end* (βλέπε Σχήμα 1.2).

Η δεύτερη προσέγγιση βασίστηκε στο γεγονός, ότι κάποια στοιχεία του διανύσματος είναι πιο σημαντικά από κάποια άλλα και επομένως ο αριθμός των *bits* που πρέπει να διαθέσουμε κυμαίνεται αρκετά, ανάλογα με τη σημαντικότητα του στοιχείου. Συγκεκριμένα, προχωρώντας από τα πρώτα στα τελευταία στοιχεία του διανύσματος, ο λόγος *bits/elements* μειώνεται δραστικά, κάτι το οποίο οφείλεται στη χρήση της *Mel-frequency* κλίμακας στο *front-end* και έχει σαν αποτέλεσμα την συμπίεση της πληροφορίας στα πρώτα στοιχεία του διανύσματος (βλέπε επίσης Σχήμα 1.2). Αυτό σημαίνει ότι προκειμένου όλα τα υποδιανύσματα να 'περιέχουν' ίσο ποσό πληροφορίας, θα πρέπει να ομαδοποιήσουμε τα πιο σημαντικά από αυτά σε μικρά υποδιανύσματα και τα λιγότερο σημαντικά σε μεγαλύτερα υποδιανύσματα. Τα αποτελέσματα για τις δυο προσεγγίσεις φαίνονται στον Πίνακα 4.4.

Bit Rate (bps)	Correlation based	Knowledge based
1400	18.77	11.71
1600	13.36	9.30
1800	10.24	8.10
1900	8.92	6.99
2000	8.38	**6.63**
2100	7.72	-
2200	7.01	-

Πίνακας 4.4: **Σύγκριση correlation & knowledge based partitioning.** Στην περίπτωση χρήσης συσχέτισης για την επιλογή των υποδιανυσμάτων, τα αποτελέσματα είναι σαφώς μη ικανοποιητικά. Τα υποδιανύσματα που χρησιμοποιήθηκαν είναι τα : (1,5), (3,9,12,13), (4,6), (2,7,11), (8,10). Στη δεύτερη περίπτωση αντίθετα, όπου το κριτήριο είναι η σημαντικότητα των στοιχείων, τα αποτελέσματα είναι σαφώς καλύτερα. Σε αυτή τη περίπτωση, τα υποδιανύσματα που χρησιμοποιήθηκαν είναι τα : (1,2), (3,4), (5,6,7), (8,9,10), (11,12,13).

4.4.3 Επιλογή αριθμού bits ανά υποδιάνυσμα : bit allocation αλγόριθμος

Η δεύτερη παράμετρος που πρέπει να καθοριστεί είναι ο αριθμός των *centroids* ανά υποδιάνυσμα. Η διαδικασία που ακολουθήσαμε βασίζεται στον *bit-allocation* αλγόριθμο και είναι η εξής :

Αρχικοποίηση : Διάθεση ενός αρχικού αριθμού *bits* στα υποδιανύσματα και αξιολόγηση της αναγνώρισης. Αυτό είναι και το αρχικό σχήμα.

Βήμα 1 : Για κάθε υποδιάνυσμα, αύξηση κατά ένα του αριθμού *bits* και νέα αξιολόγηση της αναγνώρισης, κρατώντας τον αριθμό των *bits* που έχουμε διαθέσει στα υπόλοιπα υποδιανύσματα ίσο με αυτό στο τρέχον σχήμα. Το επιπλέον *bit* το διαθέτουμε τελικά στο υποδιάνυσμα το οποίο προσέφερε την μεγαλύτερη αύξηση στην απόδοση της αναγνώρισης και θέτουμε αυτό, σαν το νέο σχήμα.

Βήμα 2 : Αν έχει επιτευχθεί το επιθυμητό αποτέλεσμα αναγνώρισης ή αν φτάσαμε το μέγιστο διαθέσιμο αριθμό *bits* σταματάμε, αλλιώς συνεχίζουμε τη διαδικασία από το Βήμα 1.

Ο ίδιος αυτός αλγόριθμος μπορεί να χρησιμοποιηθεί στην περίπτωση του μη ομοιόμορφου βαθμωτού κβαντισμού, αφού αυτός αποτελεί ειδική περίπτωση του διανυσματικού κβαντισμού, με υποδιανύσματα μήκους ενός συντελεστή. Τα αποτελέσματα φαίνονται στους πίνακες 4.5 και 4.6 αντίστοιχα.

	Composition of subVectors						
	1,2	3,4	5,6,7	8,9,10	11,12,13		
Total bits	subvector bits per iteration					Bit Rate	WER(%)
12	3	3	2	2	2	1200	16.76
14	**5**	3	2	2	2	1400	11.71
16	5	3	**4**	2	2	1600	9.30
18	5	3	4	**4**	2	1800	8.10
19	5	**4**	4	4	2	1900	6.99
20	5	**5**	4	4	2	**2000**	**6.63**

Πίνακας 4.5: **Bit allocation για τα 5 υποδιανύσματα.** Όπως φαίνεται καθαρά, τα υποδιανύσματα 1 & 2 είναι τα ποιο σημαντικά αν και μικρότερα. Ακολουθούν τα 3 & 4, ενώ το υποδιάνυσμα 5 φέρει την λιγότερη πληροφορία. Παρατηρούμε επίσης ότι με μόλις $2kbps$ διατηρούμε ίδιο *WER* με την περίπτωση όπου δεν έχουμε καν κβαντισμό!

	MFCC												
	1	2	3	4	5	6	7	8	9	10	11	12	13
Total bits	element bits per iteration													Bit Rate	*WER* (%)
17	2	2	2	2	1	1	1	1	1	1	1	1	1	1700	12.78
18	**3**	2	2	2	1	1	1	1	1	1	1	1	1	1800	10.66
20	3	**3**	2	**3**	1	1	1	1	1	1	1	1	1	2000	8.69
22	3	3	**3**	3	1	**2**	1	1	1	1	1	1	1	2200	7.67
24	3	3	3	3	**2**	2	1	1	1	1	1	**2**	1	2400	6.69
26	3	3	3	3	2	2	1	1	**2**	1	**2**	2	1	2600	6.81
28	3	3	3	3	2	2	1	**2**	2	**2**	2	2	1	2800	6.71
30	3	3	3	3	2	2	**2**	2	2	2	2	2	**2**	3000	**6.55**

Πίνακας 4.6: **Bit allocation για τη scalar περίπτωση** Φαίνεται καθαρά πως οι 4 πρώτοι συντελεστές είναι οι πιο σημαντικοί (3 *bits*) ενώ οι υπόλοιποι είναι λιγότερο σημαντικοί και απαιτούν μόνο 2 συντελεστές. Παρατηρούμε πως ήδη από τα 2600 *bits*, έχουμε καταφέρει να προσεγγίσουμε την απόδοση του *baseline* πειράματος. Παρατηρούμε επίσης πως σε σχέση με το προηγούμενο πίνακα, χρειάζονται 10 *bits* παραπάνω (*3kbps*) για να φτάσουμε την ίδια απόδοση στην αναγνώριση.

Test SNR(db)	Train VQ SNR(db)	Encoding bits/subvector	WER(%)
Clean	Clean	No encoding	6.55
Clean	Clean	5 5 4 4 2	6.63
24	Clean	No encoding	8.51
24	Clean	5 5 4 4 2	12.19
24	24	5 5 4 4 2	11.89
24	Clean	5 5 5 4 2	11.18
24	Clean	6 5 5 4 2	10.49
24	Clean	6 6 5 4 3	9.47
24	Clean	7 6 5 4 4	9.32
24	Clean	7 6 5 4 5	8.94

Πίνακας 4.7: **Κβαντισμός παρουσία θορύβου.**

4.4.4 Πειράματα με θόρυβο.

Στην εφαρμογή της μεθόδου που ακολουθήσαμε έως τώρα, δεν υπολογίσαμε την επίδραση του θορύβου. Για να διαπιστώσουμε την επίδραση του θορύβου, κάναμε το εξής πείραμα :Προστέθηκε θόρυβος στο *test set* έτσι ώστε ο λόγος σήματος/θορύβου (*SNR*) να φτάσει τη τιμή των 24 *dB*. Ακολουθώντας το *bit allocation* σχήμα για άλλη μια φορά, προσπαθήσαμε να βρούμε το κατάλληλο αριθμό *bits* που απαιτούνται για να φτάσουμε τα επίπεδα αναγνώρισης στη περίπτωση του μη κβαντισμού. Τα αποτελέσματα φαίνονται στον Πίνακα 4.7.

Αναλυτικότερα, στη 1η και 2η γραμμή επαναλαμβάνονται τα αποτελέσματα που πήραμε πριν την εισαγωγή του θορύβου για τα πειράματα χωρίς κβαντισμό (*baseline*) και για το καλύτερο πείραμα με χρήση κβαντισμού. Όλα τα επόμενα πειράματα είναι παρουσία θορύβου. Στην 3η και 4η γραμμή επαναλαμβάνουμε τα 2 παραπάνω πειράματα . Το σφάλμα αναγνώρισης χωρίς κβαντισμό είναι *8.51%* ενώ με κβαντισμό αυξάνει ακόμα περισσότερο φτάνοντας το *12.19%*. Αν προσθέσουμε 24 *dB* στο *train set* έχουμε μικρή μείωση του σφάλματος (*11.89%*). Στα επόμενα πειράματα εφαρμόζουμε το *bit-allocation* σχήμα με σκοπό να προσεγγίσουμε το *8.51%*. Στις υπόλοιπες γραμμές φαίνεται πως μειώνεται το *WER* στο σε *8.94* προσθέτοντας συνολικά 7 *bits* παραπάνω, φτάνοντας έτσι στα 27 *bits*, δηλαδή *2.7kbps*.

4.5 Συμπεράσματα

- Η διαδικασία πειραμάτων αποτελείται από τις διαδικασίες εκπαίδευσης, κβαντισμού και αναγνώρισης. Τα αποτελέσματα της αναγνώρισης (έχου-

με κωδικοποίηση φωνής προς αναγνώριση) είναι αυτά που κρίνουν την ποιότητα της κωδικοποίησης.

- Το *baseline* πείραμα μας δίνει αποτελέσματα αναγνώρισης χωρίς κβαντισμό, τα οποία και θα προσπαθήσουμε να προσεγγίσουμε και στα πειράματα της κωδικοποίησης.

- Ο βαθμωτός κβαντισμός αν και απλός στην υλοποίηση δίνει αρκετά καλά αποτελέσματα. Ο μη-ομοιόμορφος υπερέχει του ομοιόμορφου βαθμωτού κβαντισμού.

- Για τον διανυσματικό κβαντισμό πρέπει να κάνουμε επιλογές σχετικά με το πως θα χωρίσουμε το *MFCC* διάνυσμα σε μικρότερα υποδιανύσματα και με το πόσα *bits* θα διαθέσουμε για την εκπαίδευσή τους. Για την πρώτη επιλογή χρησιμοποιήσαμε την γνώση μας σχετικά με την σημαντικότητα των συντελεστών του *MFCC* διανύσματος ενώ για την δεύτερη τον *bit-allocation* αλγόριθμο.

- Τα αποτελέσματα των πειραμάτων διανυσματικού κβαντισμού είναι εξαιρετικά . Με *bit-rate* μόλις **2Kbps** παίρνουμε ακρίβεια αναγνώρισης ίση, με την περίπτωση του μη κβαντισμού(*baseline* πείραμα), ενώ και στη περίπτωση θορύβου, το σύστημα αποδείχθηκε αρκετά εύρωστο απαιτώντας επιπρόσθετα μόλις $0.7Kbps$. Τα αποτελέσματα αυτά δημοσιεύτηκαν στα [10], [11], και [12].

Κεφάλαιο 5

Χρήση κωδικοποίησης για αναγνώριση σε δίκτυα δεδομένων

5.1 Εισαγωγή

Στο Εισαγωγικό Κεφάλαιο αναφερθήκαμε στις ραγδαίες εξελίξεις στο χώρο των δικτύων και τον ασύρματων επικοινωνιών και προτείναμε τη χρησιμοποίηση αυτών των νέων τεχνολογιών για την περαιτέρω εξάπλωση εφαρμογών αναγνώρισης φωνής. Στο πρώτο μέρος αυτού του κεφαλαίου θα εξετάσουμε διεξοδικά τις εξελίξεις στον τομέα των δικτύων (με έμφαση στις ασύρματες επικοινωνίες) και θα δούμε πως μπορούμε να χρησιμοποιήσουμε εφαρμογές αναγνώρισης φωνής σε αυτό το περιβάλλον, βασιζόμενοι στην κωδικοποίηση που αναπτύξαμε σε αυτή την εργασία. Η κωδικοποίηση που αναπτύξαμε παίζει καταλυτικό παράγοντα, μειώνοντας στο ελάχιστο τις απαιτήσεις σε ρυθμό μετάδοσης δεδομένων καθιστώντας έτσι δυνατή την ανάπτυξη εφαρμογών σε δίκτυα δεδομένων με μικρό κόστος.

Αναφέραμε επίσης στο Εισαγωγικό Κεφάλαιο, ότι τέτοιες εφαρμογές εκ των πραγμάτων θα χρησιμοποιούν το μοντέλο πελάτη - εξυπηρετητή για αναγνώριση φωνής. Στο δεύτερο μέρος του κεφαλαίου, περιγράφεται ένα παράδειγμα μιας τέτοιας υλοποίησης για αναγνώριση φωνής σε δίκτυο μέσα από ένα δίαυλο *2kbps*. Ο πελάτης χρησιμοποιεί το σύστημα *Yarrow* που εξετάσαμε στο Κεφάλαιο 3 και ο εξυπηρετητής το σύστημα αναγνώρισης φωνής *Decipher* κατάλληλα προσαρμοσμένο για συνεργασία με το *Yarrow*. Επίσης θα εξετάσουμε την αρχιτεκτονική του πελάτη και τον τρόπο επικοινωνίας με τον αναγνωριστή. Θα αρχίσουμε αυτό το κεφάλαιο εξετάζοντας πρώτα τις εφαρμογές αναγνώρισης φωνής που κυριαρχούν σήμερα.

5.2 Εφαρμογές αναγνώρισης φωνής σήμερα

Παραδείγματα εφαρμογών αναγνώρισης φωνής είναι :

- Εφαρμογές γραφείου όπως συστήματα υπαγόρευσης, εισαγωγή δεδομένων.
- Παρακολούθηση της κατασκευαστικής διαδικασίας και έλεγχος ποιότητας.
- Έλεγχος με φωνή, συστήματα για άτομα με ειδικές ανάγκες, εφαρμογές όπως η ρύθμιση του κλιματισμού ή του συστήματος ήχου σε ένα αυτοκίνητο.
- Εφαρμογές στην ιατρική όπως για παράδειγμα έλεγχος ρομποτικού βραχίονα για επεμβάσεις από απόσταση.
- Εφαρμογές στις τηλεπικοινωνίες και τηλεφωνικά κέντρα. Αυτοματοποίηση χειρισμού τηλεφωνικών κέντρων, πρόσβαση σε πληροφορίες.

Την μεγαλύτερη απήχηση από τις παραπάνω, έχουν οι εφαρμογές αναγνώρισης φωνής για ανάκτηση πληροφοριών με τη χρήση τηλεφώνου. Η χρήση μέσω ενός απλού τηλεφώνου κάνει τις υπηρεσίες αυτές προσβάσιμες σε οποιονδήποτε. Οι πληροφορίες αφορούν για παράδειγμα εύρεση εισιτηρίων για αεροπορικές πτήσεις, αγορά προϊόντων, μετοχών κ.τ.λ. Μόνο στις Ηνωμένες Πολιτείες τέτοιες εφαρμογές δέχονται καθημερινά αρκετές δεκάδες χιλιάδων τηλεφωνικών κλήσεων. Τέτοιες υπηρεσίες είναι βέβαια προσβάσιμες και από κινητά τηλέφωνα κάτι που αποτελεί μεγάλο πλεονέκτημα καθώς κάνει την πρόσβαση σε αυτές τις υπηρεσίες δυνατή από οπουδήποτε.

Η ανάκτηση πληροφοριών μπορεί να γίνεται είτε από συγκεκριμένες βάσεις δεδομένων είτε χρησιμοποιώντας τον πλούτο των πληροφοριών που υπάρχει στο διαδίκτυο μέσω του προτύπου της *VoiceXML*. Η *VoiceXML* είναι μια τεχνολογία που βασίζεται στην *XML* και που επιτρέπει να γίνει προσβάσιμο το περιεχόμενο δικτυακών τόπων μέσω φωνής, απαιτεί όμως για κάτι τέτοιο την δημιουργία ξεχωριστών δικτυακών τόπων που ονομάζονται φωνητικές πύλες (*voice portals*).

5.3 Ασύρματα δίκτυα και συσκευές νέας γενεάς

Η ταχεία ανάπτυξη των ασύρματων δικτύων και η εξάπλωση τις κινητής τηλεφωνίας, μαζί με την μεγάλη ανάπτυξη που υφίσταται η αγορά *embedded* συστημάτων, *consumer electronics* και μιας πληθώρας άλλων συσκευών από

palmtops και *pdas* έως *smartphones*, δείχνει ότι οδεύουμε στην μετά *PC* εποχή στην οποία η αναζήτηση και επεξεργασία πληροφοριών, θα γίνεται από πολλές διαφορετικού τύπου τέτοιες συσκευές. Αν λοιπόν η πρόσβαση στο διαδίκτυο από τον προσωπικό υπολογιστή έχει γίνει τόσο εύκολη και απαραίτητη, γιατί να μην ισχύει το ίδιο και για αυτές τις συσκευές, που ήδη σήμερα θεωρούνται απαραίτητα εργαλεία, όπως ένα *pda* ή ένα κινητό τηλέφωνο;

Στη σημερινή κινητή τηλεφωνία, ήδη υπάρχει υποστήριξη δεδομένων (*cicruit-switched* όπως *HSCSD*) και υπηρεσίες όπως *SMS & WAP*. Στο δρόμο προς τη κινητή τηλεφωνία 3ης γενεάς μας οδηγούν *packet-switched* πρωτόκολλα όπως τα *GPRS, EDGE, WCDMA* με βασικό χαρακτηριστικό όλων, την πλήρη υποστήριξη δικτύων δεδομένων *IP* σε σταδιακά όλο και υψηλότερες ταχύτητες μεταφοράς δεδομένων(*broadband networks*). Έτσι, σταδιακά ένας πολύ μεγάλος αριθμός κινητών συσκευών (πολύ μεγαλύτερος από τον αριθμό προσωπικών υπολογιστών με σύνδεση στο δίκτυο) θα έχει πρόσβαση στο λεγόμενο *mobile internet* του αύριο.

5.3.1 Οφέλη από τη μετάβαση σε ασύρματα δίκτυα δεδομένων

Το *GPRS* (βλέπε [13]) τέθηκε σε εφαρμογή στο τέλος του 2001 εγκαινιάζοντας την $2.5G$ γενεά κινητής τηλεφωνίας (η οποία βασίζεται στα ήδη υπάρχοντα δίκτυα με κύριες αλλαγές σε λογισμικό), ενώ ήδη οι περισσότεροι παροχείς κινητής τηλεφωνίας έχουν καταβάλλει υπέρογκα ποσά για άδειες της επερχόμενης 3η γενεάς, ελπίζοντας έτσι να εξασφαλίσουν ένα μερίδιο στην νέα αγορά που γεννιέται. Τα κέρδη από τη μετάβαση στα δίκτυα αυτά θα είναι μεγάλα, τόσο για τους παροχείς αυτών των δικτύων αλλά και για τον χρήστη αυτής της νέας τεχνολογίας.

Για τους παροχείς, δίνεται η ευκαιρία να εκμεταλλευτούν καλύτερα την χωρητικότητα του δικτύου, λόγω της *packet switched* τεχνολογίας (σε αντίθεση με τη σημερινή *circuit switched*). Αυτό σημαίνει ότι τα *time slots* δεσμεύονται δυναμικά για τη μεταφορά δεδομένων και όχι στατικά, ανεξάρτητα από την ανάγκη για *bandwidth*. Αυτό βέβαια αποτελεί μεγάλο πλεονέκτημα και για το χρήστη, αφού η χρέωση θα είναι ανά πακέτο πληροφορίας και όχι ανά χρόνο σύνδεσης (κάτι τέτοιο άλλωστε δεν θα υφίσταται αφού θα είναι συνεχώς συνδεμένος στο δίκτυο του παροχέα).

Ο κάτοχος μιας κινητής συσκευής, θα μπορεί να έχει πρόσβαση στο διαδίκτυο ή στο εταιρικό εσωτερικό δίκτυο (*corporate intranet*) από οπουδήποτε. Τα κινητά τηλέφωνα θα εξελίσσονται διαρκώς και θα είναι δυνατό ο χρήστης να εγκαθιστά κατάλληλο λογισμικό για κάποια υπηρεσία. Αυτός είναι και ο λόγος που αναμένεται μεγάλο μερίδιο της μελλοντικής αυτής αγοράς να δεσμευτεί από υπηρεσίες παροχής εφαρμογών (*ASPs*). Παρ'όλες όμως τις δυνατότητες

που θα παρέχουν τα κινητά ή άλλες παρόμοιες συσκευές, η αλληλεπίδραση θα παραμένει ακόμα σε χαμηλά επίπεδα (οθόνη 5 γραμμών ή με μικρή ανάλυση). Σε κάτι τέτοιο μπορεί να βοηθήσει η αναγνώριση φωνής με τα συγκριτικά πλεονεκτήματα που προσφέρει.

5.4 Μοντέλο κατανεμημένης αναγνώρισης φωνής

Με την έλευση δικτύων όπως αυτά της 3ης γενεάς, αναμένεται αργά η γρήγορα η κυριαρχία των δικτύων δεδομένων έναντι των δικτύων φωνής (τεχνολογίες όπως *Voice Over IP (VoIP)* και μετάβαση σε *all-IP* δίκτυα). Με βάση λοιπόν τη παράγραφο 5.1, όπου είδαμε ότι η πιο διαδεδομένη εφαρμογή αναγνώρισης φωνής, είναι η χρησιμοποίηση του σημερινού ενσύρματου και ασύρματου τηλεφωνικού δικτύου για ανάκτηση πληροφοριών κάθε είδους [1], θα δούμε πως μπορεί να επεκταθεί μια τέτοια εφαρμογή/υπηρεσία για να γίνει διαθέσιμη και από συσκευές με πρόσβαση σε ασύρματα δίκτυα δεδομένων.

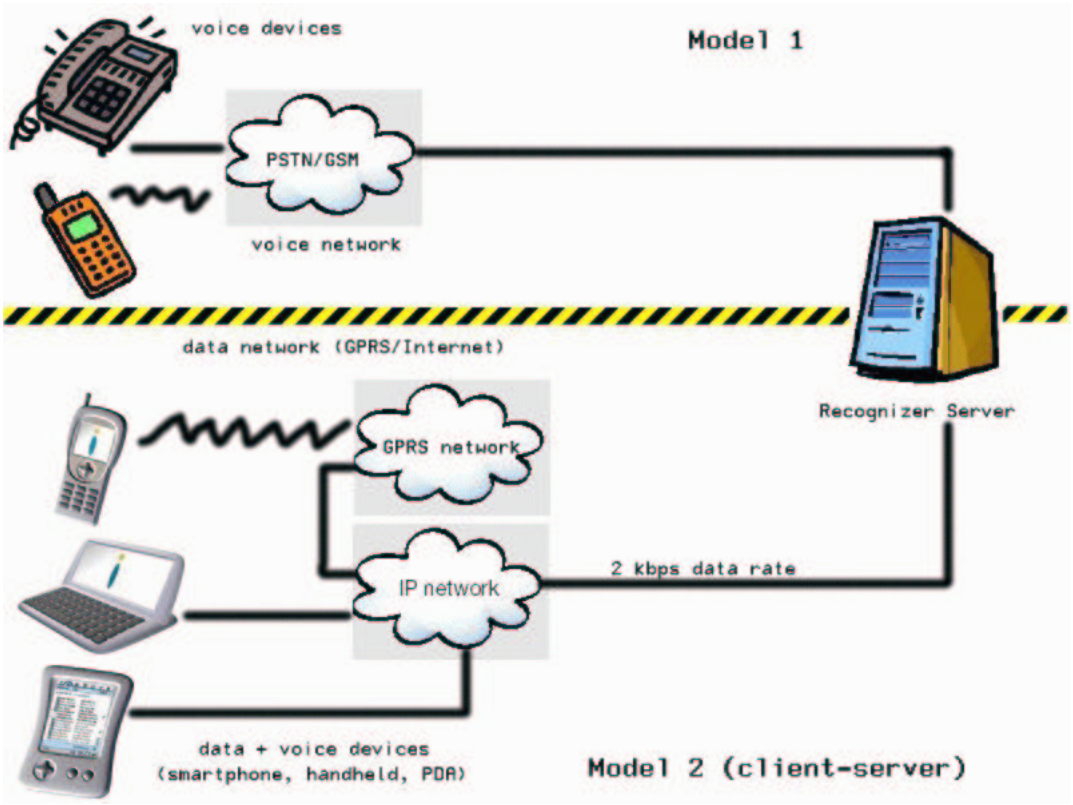
Τα δυο μοντέλα πρόσβασης σε υπηρεσίες αναγνώρισης φωνής μέσω δικτύων συγκρίνονται στο Σχήμα 5.1. Το πρώτο μοντέλο χρησιμοποιεί συσκευές και δίκτυα φωνής ενώ το δεύτερο συσκευές με πρόσβαση σε ασύρματα δίκτυα δεδομένων όπως π.χ. *GPRS* ή *802.11*. Στο δεύτερο μοντέλο γίνεται η χρήση της κωδικοποίησης στις συσκευές αυτές, με αποτέλεσμα να απαιτούνται μόλις $2Kbps$ για τη μετάδοση της κωδικοποιημένης φωνής στον αναγνωριστή όπου και συντελείται το υπόλοιπο μέρος της αναγνώρισης. Επειδή μέρος της αναγνώρισης (*front-end* επεξεργασία και κβαντισμός) λαμβάνει χώρα στις συσκευές αυτές, το μοντέλο αυτό πελάτη - εξυπηρετητή συχνά αναφέραται και ως κατανεμημένη αναγνώριση φωνής.

5.4.1 Σχετικά πρότυπα : ETSI proposal

Το *ETSI* έχει συστήσει την ομάδα εργασίας *STQ-Aurora DSR*, με σκοπό την δημιουργία ενός στάνταρ για ανάπτυξη κατανεμημένων εφαρμογών αναγνώρισης φωνής, για πρόσβαση από κινητές συσκευές. Στην ομάδα εργασίας συμετέχουν κυρίως εταιρίες κατασκευής κινητών τηλεφώνων (*Nokia, Motorola, Ericsson*) όχι όμως εταιρίες συστημάτων αναγνώρισης φωνής, κάτι που κάνει την υιοθέτησή του αμφίβολη, τουλάχιστον προς το παρόν.

Το στάνταρ προτείνει φυσικά το μοντέλο πελάτη-εξυπηρετητή, όπως άλλωστε και το δικό μας μοντέλο και δίνει έμφαση κυρίως στην *front-end* επεξερ-

[1] Ο χρήστης καλεί από το τηλέφωνό κάποιο καθορισμένο αριθμό ζητώντας κάποια πληροφορία, η φωνή μεταδίδεται σε μορφή *m-law* (σταθερή τηλεφωνία) ή *GSM* (κινητή τηλεφωνία) και αφού αναγνωριστεί η πρόταση, το σύστημα απαντάει χρησιμοποιώντας *text to speech synthesis*



Σχήμα 5.1: Αναγνώριση φωνής σε δίκτυα φωνής και δεδομένων

γασία. Σύμφωνα με το στάνταρ, το *front-end* παράγει ακουστικά διανύσματα μεγέθους 14 στοιχείων (13 *cepstral* παράμετροι *C0-C12* και *logE*) τα οποία κβαντίζονται με διανυσματικό κβαντισμό (6 *bits* για κάθε ζεύγος από *C1-C12* και 8 *bits* για *C0-logE*, σύμφωνα με πρόταση της *Motorola*). Προκύπτει έτσι κωδικοποίηση με ρυθμό δεδομένων *4.4Kbps*, πολύ μεγαλύτερο από το *2Kbps* που πετύχαμε σε αυτή την εργασία. Για περισσότερες πληροφορίες παραπέμπουμε στην [14].

5.4.2 Τεχνολογία ασύρματων συσκευών πληροφορίας

Με τον παραπάνω όρο (*Wireless/Mobile Information Devices*) αναφερόμαστε σε συσκευές όπως *high-end mobiles* & pdas με πρόσβαση σε ασύρματα δίκτυα δεδομένων από *Bluetooth* & 802.11 έως *GPRS*. Βέβαια η αγορά των κινητών τηλεφώνων είναι πολύ μεγαλύτερη από αυτή των *pdas*, γι' αυτό συχνά θα αναφερόμαστε στον όρο τηλεφώνα νέας γενεάς.

Κατά γενική ομολογία, οι δυνατότητές τους θα αυξηθούν κατακόρυφα στο σύντομο μέλλον, με δυο κύριες υποκατηγορίες : τα έξυπνα τηλέφωνα (*smart-phones*) φωνο-κεντρικές (*voice-centric*) συσκευές με δυνατότητες επεξεργασίας & ανταλλαγής πληροφορίας (π.χ. το *Nokia 7650*) και *communicators*, πληροφοριο-κεντρικές (*information-centric*) συσκευές με δυνατότητες φωνής (π.χ. *Nokia 9310*) και εισόδου από πληκτρολόγιο, στυλό ή άλλο τρόπο, όπως φωνή (*multimodal interfaces*). Οι συσκευές αυτές, έχουν συνήθως επεξεργαστές με πολύ μικρή κατανάλωση ισχύος και με ταχύτητες που φτάνουν αυτή τη στιγμή έως και τα $206MHz$ όπως π.χ. ο *StrongArm* επεξεργαστής στο *Ericsson P800*. Τρέχουν ειδικά σχεδιασμένα λειτουργικά συστήματα σαν το *Symbian* (κοινοπραξία *Ericsson, Nokia, Motorola*) με βιβλιοθήκες ανάπτυξης προγραμμάτων σε γλώσσες όπως *C++, Java*.

Ειδικά για ανάπτυξη με τη χρήση *Java*, υπάρχει ειδική έκδοση της πλατφόρμας, με την κωδική ονομασία *J2ME* (*Java 2 Micro Edition*, βλέπε [15]). Η *J2ME* παρέχει δυο διαφορετικά *configurations* που απευθύνονται σε δυο διαφορετικές κατηγορίες συσκευών : τα *Connected & Connected Limited Device Configurations - (CDC & CLDC* αντίστοιχα). Εκτός από τα *configurations* που καθορίζουν το είδος της *virtual machine* (*CVM & KVM* αντίστοιχα) υπάρχουν και τα λεγόμενα *profiles* που προσφέρουν επιπρόσθετες βιβλιοθήκες για μια συγκεκριμένη υποκατηγορία συσκευών. Το *CDC* αποτελεί τον αντικαταστάστη της *Personal Java* σε συσκευές όπως *high-end pdas* ενώ το *CLDC* με το *Mobile Information Devices Profile - MIDP* χρησιμοποιείται στα περισσότερα νέα κινητά.

5.4.3 Θέματα χρήσης σε δίκτυα νέας γενεάς

Παρά τον ενθουσιασμό που υπάρχει γύρω από τις υπηρεσίες μεγάλου εύρους ζώνης που θα έρθουν με τα δίκτυα 3ης γενεάς, κανείς πρέπει να κρατάει κάποιες επιφυλάξεις. Ο λόγος είναι ότι οι ρυθμοί μετάδοσης που ανακοινώνονται για τα πρωτόκολλα αυτά, είναι το θεωρητικό μέγιστο. Για παράδειγμα, για το *GPRS* μέγιστος θεωρητικός ρυθμός δεδομένων είναι τα $171.2kbps$ (κάτι που απαιτεί δέσμευση 8 *time slots* και χρήση κωδικοποίησης *CS-4*), στην πράξη όμως δεν αναμένεται να ξεπεράσει τα $50kbps$. Τα επόμενα πρωτόκολλα μέχρι τα 3ης γενεάς, θα υποστηρίζουν θεωρητικά μέχρι και $2Mbps$, ποτέ όμως δεν πρόκειται να δούμε κάτι τέτοιο στη πράξη. Επίσης, όσο πιο εξελιγμένα είναι τα πρωτόκολλα αυτά, τόσο περιορισμένα αναμένεται να είναι από άποψη κάλυψης : για παράδειγμα το *EDGE* δεν αναμένεται να καλύψει περιοχές εκτός πόλεων.

Βέβαια το πιο σημαντικό στοιχείο των δικτύων αυτών είναι όπως ήδη αναφέραμε η υποστήριξη του *IP* πρωτοκόλλου, κάτι που κάνει δυνατή τη μεταφορά εφαρμογών του διαδικτύου. Κάτι τέτοιο δεν είναι πάντα εύκολο, αφού το

ασύρματο κανάλι έχει εντελώς διαφορετική συμπεριφορά από ένα ενσύρματο. Τα δυο βασικότερα χαρακτηριστικά που το διαφοροποιούν, είναι το υψηλότερο επίπεδο θορύβου από τη μια και η πολύ μεγαλύτερη καθυστέρηση (*latency*) των πακέτων από τη άλλη. Έτσι π.χ. από τα συστήματα πελάτη εξυπηρετητή ευνοούνται αυτά με μικρό αριθμό ανταλλαγής πληροφορίας (μικρό αριθμό αιτήσεων/απαντήσεων), όπως η εφαρμογή μας, ενώ αντίθετα π.χ. η πλοήγηση σε μια σελίδα απαιτεί πολύ μεγάλο αριθμό τέτοιων αιτήσεων.

Βλέπουμε επομένως, ότι η ανάπτυξη εφαρμογών για πρόσβαση από ασύρματο περιβάλλον, απαιτεί να συμπεριληφθούν υπόψη διάφοροι παράγοντες που έχουν να κάνουν τόσο με τα ίδια τα δίκτυα, όσο και με τις συσκευές που παρέχουν πρόσβαση σε αυτά. Προκειμένου να υποστηριχθεί περαιτέρω η ανάπτυξη τέτοιων εφαρμογών, έχει ιδρυθεί ανάλογη πρωτοβουλία (*Mobile Applications Initiative*) (βλέπε [17]). Επίσης πρέπει να γίνει κατανοητό, ότι δεν είναι όλες οι εφαρμογές κατάλληλες για ένα τέτοιο περιβάλλον, άλλες για τεχνικούς λόγους και άλλες για λόγους χρησιμότητας. Αναμένεται τέτοιες εφαρμογές, να είναι απλές από άποψη χρήσης και να παρέχουν γρήγορη και εύκολη ανάκτηση, μικρών σε ποσότητα πληροφοριών, στο πλαίσιο στο οποίο ανήκει άλλωστε και η δική μας εφαρμογή (βλέπε [17]).

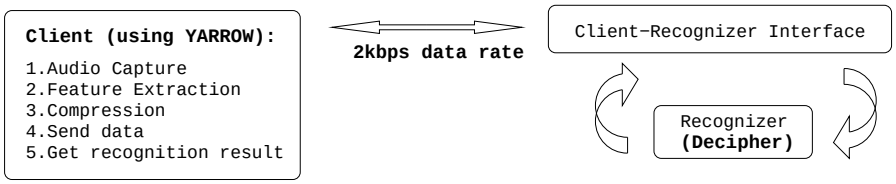
5.5 Παράδειγμα εφαρμογής κατανεμημένης αναγνώρισης φωνής

Σε αυτή την ενότητα θα εξετάσουμε τη δομή του συστήματος που αναπτύξαμε για αναγνώριση φωνής σε δίκτυο δεδομένων. Πρέπει να σημειωθεί σε αυτό το σημείο, ότι η υλοποίηση που παρουσιάζεται εδώ είναι απλά ένα τέτοιο παράδειγμα που σκοπό έχει να καταδείξει τον τρόπο χρήσης της κωδικοποίησης στο μοντέλο πελάτη-εξυπηρετητή. Ωστόσο το παράδειγμα αυτό δείχνει τον τρόπο που θα μπορούσε να χτιστεί μια πραγματική εφαρμογή/υπηρεσία που θα χρησιμοποιεί την κωδικοποίηση φωνής που αναπτύχθηκε σε αυτή την εργασία όπως π.χ. η ανάκτηση πληροφοριών μέσω δικτύου δεδομένων με χρήση φωνής (σε αναλογία με τις αντίστοιχες υπηρεσίες ανάκτησης πληροφοριών μέσω δικτύου φωνής της παράγραφου 5.1).

Σε μια τέτοια περίπτωση τα πλεονεκτήματα χρήσης για ένα παροχέα μιας τέτοιας υπηρεσίας θα ήταν σημαντικά :

- Καθίσταται μη αναγκαία η ύπαρξη εξοπλισμού τηλεφωνικού κέντρου και διανομής κλήσεων, κάτι που απαιτεί επιπρόσθετο κόστος από τη μια, αλλά και εξειδικευμένες λύσεις σε υλικό και λογισμικό από την άλλη.

- Οι ελάχιστες απαιτήσεις σε ρυθμό μετάδοσης δεδομένων (*2kbps*) επιτρέ-



Σχήμα 5.2: Χρήση κωδικοποίησης στο μοντέλο πελάτη - εξυπηρετητή

πει σε τέτοια συστήματα και υπηρεσίες να υποστηρίζουν πολύ μεγάλο αριθμό συνδέσεων με αναγνωριστές (υποστήριξη μεγάλης κλίμακας - *large scale support*). Αντίθετα, κάτι τέτοιο είναι πολύ δύσκολο να γίνει στην περίπτωση χρήσης τηλεφωνικού δικτύου, αφού υπάρχουν περιορισμοί ως προς τον μέγιστο αριθμό τηλεφωνικών καναλιών και ακόμα επιπρόσθετο κόστος.

5.5.1 Αρχιτεκτονική του συστήματος

Η εφαρμογή μας ακολουθεί το μοντέλο πελάτη - εξυπηρετητή (*client - server*), όπως ήδη αναφέραμε. Ο πελάτης είναι ένα εφαρμογίδιο (*applet*),[2] που αιχμαλωτίζει το σήμα της φωνής, την κωδικοποιεί σε $2Kbps$ χρησιμοποιώντας το σύστημα *Yarrow* και τη στέλνει μέσω δικτύου στον εξυπηρετητή προς αναγνώριση. Ο αναγνωριστής (*Decipher*) επιτελεί όλο το δύσκολο έργο της αναγνώρισης και ενημερώνει τον πελάτη για το αποτέλεσμα. Η γενική μορφή του συστήματος φαίνεται στο Σχήμα 5.2

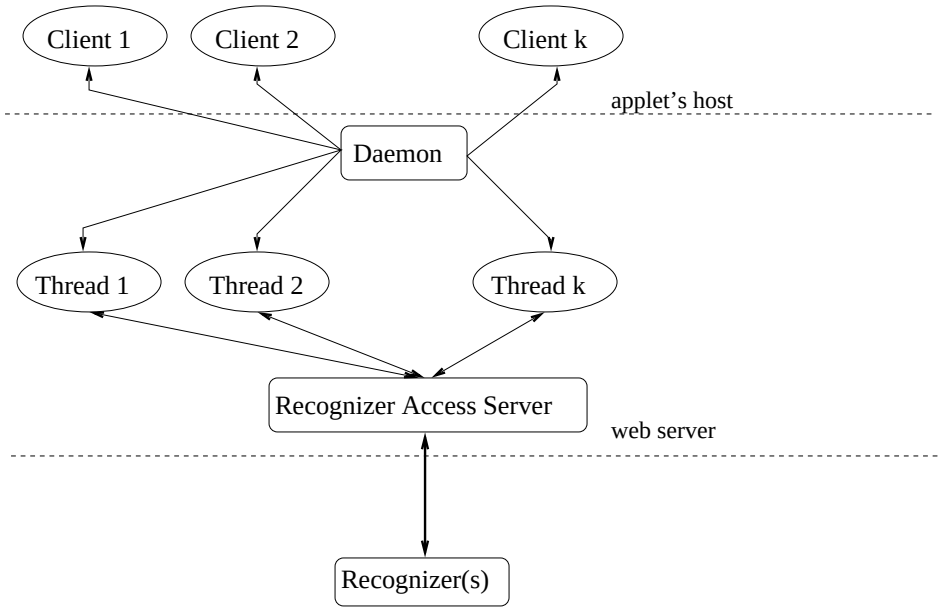
Όπως φαίνεται στο σχήμα, το σύστημα αποτελείται από 3 μέρη : Τον πελάτη(*applet*), τον εξυπηρετητή(αναγνωριστή) καθώς και από το συνδετικό τους κρίκο *(Client Recognizer Interface - CRI)*.

5.5.2 Επικοινωνία πελάτη αναγνωριστή

Η επικοινωνία των πελατών με τον αναγνωριστή πρέπει να υποστηρίζει την εξυπηρέτηση πολλών πελατών ταυτόχρονα. Μια υλοποίηση θα ήταν να ενσωματωθεί αυτή η ικανότητα απευθείας στον κώδικα του αναγνωριστή έτσι ώστε για κάθε πελάτη, ο αναγνωριστής να δημιουργεί ένα καινούριο *process* για να τον εξυπηρετήσει. Για λόγους υλοποίησης όμως κυρίως[3], αποφασίστηκε να

[2]Η διαφορά *Java applet* και *application* απλά έχει να κάνει με το περιβάλλον στο οποίο τρέχει ο κώδικας, αλλά η λειτουργικότητα είναι η ίδια. Στην υλοποίηση υπάρχουν και οι 2 μορφές αλλά σαν σύμβαση από εδώ και πέρα θα χρησιμοποιείται ο όρος *applet*

[3]Ένα *applet* μπορεί να επικοινωνεί μόνο με την *IP* διεύθυνση του *web server*, κάτι που σημαίνει ότι και ο αναγνωριστής θα έπρεπε να τρέχει στο ίδιο μηχάνημα!



Σχήμα 5.3: Επικοινωνία πελάτη αναγνωριστή μέσω *daemon, daemon threads & RAS* για μια συγκεκριμένη εφαρμογή-υπηρεσία

ακολουθηθεί η δομή του Σχήματος 5.3.

Οι πελάτες επικοινωνούν με τον αναγνωριστή μέσω του *daemon* ο οποίος δημιουργεί ένα *thread* για κάθε ένα πελάτη, το οποίο με τη σειρά του αναλαμβάνει να τον εξυπηρετήσει παρέχοντάς του πρόσβαση στον αναγνωριστή. Αφού το *thread* κερδίσει πρόσβαση στον αναγνωριστή με βάση το *id* του μέσω του *Recognizer Access Server (RAS)*, θα επικοινωνήσει με τον αναγνωριστή. Ο αναγνωριστής θα διαβάσει τα δεδομένα (κωδικοποιημένη φωνή) και θα επιστρέψει την πρόταση που αναγνώρισε στο *thread*, το οποίο με τη σειρά του θα την επιστρέψει στον πελάτη.

Η υλοποίηση αυτή, υποθέτει ότι υπάρχει διαθέσιμος ένας μόνο αναγνωριστής, ο οποίος μάλιστα εξυπηρετεί σειριακά τις αιτήσεις για αναγνώριση. Σε ένα άλλο σενάριο (π.χ. μια πραγματική υπηρεσία), θα μπορούσε να υπάρχει μια φάρμα τέτοιων αναγνωριστών χωρισμένη σε γκρουπς, καθένα από τα οποία αντιστοιχεί σε μια καθορισμένη υπηρεσία ή εφαρμογή π.χ. ένα γκρουπ με αναγνωριστές για κράτηση αεροπορικών εισιτηρίων, ένα άλλο για πληροφορίες κίνησης μετοχών κ.ο.κ. Σε μια τέτοια περίπτωση, για κάθε τέτοια υπηρεσία θα υπάρχει και ένας διαφορετικός *daemon*, ενώ ο αντίστοιχος *RAS* θα εξυπηρετεί ένα τέτοιο γκρουπ αναγνωριστών ενεργόντας έτσι και σαν μηχανισμός

εξισορρόπησης φορτίου (*load balancing mechanism*).

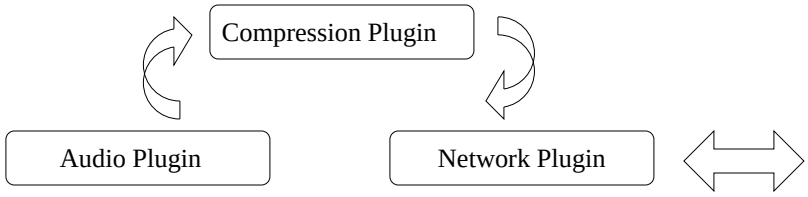
5.5.3 Υλοποίηση του πελάτη για batch αναγνώριση

Σε αυτή την υλοποίηση του πελάτη, σκοπός είναι να δειχθεί η ακολουθία των βημάτων για να γίνει η αναγνώριση. Τα βήματα αυτά γίνονται σειριακά, δηλαδή το επόμενο αρχίζει αφού τελειώσει το προηγούμενο και γι' αυτό αναφερόμαστε στον όρο *batch* αναγνώριση. Στο επόμενο κεφάλαιο θα δούμε πως μπορεί να γίνει *live* αναγνώριση.

Ο πελάτης εκτελεί σειριακά τις ακόλουθες λειτουργίες :

- Ηχογράφηση και αναπαραγωγή της προς αναγνώριση πρότασης (*Audio capture*)
- Εξαγωγή και συμπίεση των παραμέτρων της φωνής (*Feature extraction & Compression*)
- Αποστολή της κωδικοποιημένης φωνής στον αναγνωριστή και ενημέρωση για το αποτέλεσμα της αναγνώρισης (*Recognition*)

Για κάθε μια από τις παραπάνω λειτουργίες υπάρχει ένα *software module (plugin)* που είναι υπεύθυνο για να τις εκτελέσει. Τα 3 *plugins* είναι (βλέπε Σχήμα 5.4) : το *Audio (AP)*, το *Compression (CP)* και το *Network (NP)*.



Σχήμα 5.4: Τα 3 plugins του πελάτη.

Audio Plugin

Το *Audio Plugin* είναι υπεύθυνο για την ηχογράφηση της φωνής. Επειδή η *Java* είναι μια γλώσσα προγραμματισμού ανεξάρτητη πλατφόρμας, δεν παρέχει πρόσβαση σε λειτουργίες εξαρτώμενες από κάποια συγκεκριμένη πλατφόρμα (*platform independed language*). Για να γίνει κάτι τέτοιο θα πρέπει να γίνει ο προγραμματισμός για αυτές τις λειτουργίες σε κάποια άλλη γλώσσα προγραμματισμού και στη συνέχεια η σύνδεση του κώδικα αυτού με την ιδεατή

μηχανή (*virtual machine*) της *Java* μέσω ενός πρωτοκόλλου που ονομάζεται *Java Native Interface - (JNI)*.

Ο *platform dependent* κώδικας γράφτηκε σε *C* και μεταγλωττίστηκε σε βιβλιοθήκη, η οποία φορτώνεται από τον *java* κώδικα του *audio plugin* σε *run-time* για να επιτελέσει τις ζητούμενες λειτουργίες *(recording/rendering)*. Η πλατφόρμα στην οποία έγινε η υλοποίηση είναι το λειτουργικό σύστημα *Linux*. Παρόλα αυτά, ο *audio driver*[4] που χρησιμοποιήθηκε, υπάρχει διαθέσιμος σε όλες τις πλατφόρμες *UNIX*.

Compression Plugin

Το *plugin* αυτό χρησιμοποιεί 3 *clients* του κεφαλαίου 3 (βλέπε *package sri.star .coding.clients*) για να επιτελέσει τη κωδικοποίηση. Είναι το πλέον βασικό *plugin*, που χρησιμοποιεί όλη τη τεχνολογία που αναπτύχθηκε για αυτή την εργασία.

- Ο *feature producer client (fpc)* είναι υπεύθυνος για να εφαρμόσει *front-end* ανάλυση στο σήμα της κυματομορφής που παίρνει από το *AudioPlugin*. Το αποτέλεσμα είναι ένας πίνακας MxN όπου N είναι το μέγεθος του *feature* που παράγουμε (π.χ. 13 για *Cepstrum*) και M ο αριθμός των *frames*.
- Ο *quantizer client (qc)* κβαντίζει το *feature* χρησιμοποιώντας ένα σχήμα κωδικοποίησης όπως αυτό των *2 kbps*.
- Ο *ascii gziped writer (agw)* δημιουργεί μια συμπιεσμένη αναπαράσταση του κβαντισμένου *feature* την οποία αργότερα θα γράψει στο *socket* 'προς' τον αναγνωριστή (το οποίο δημιουργείται από το *Network Plugin*).

Network Plugin

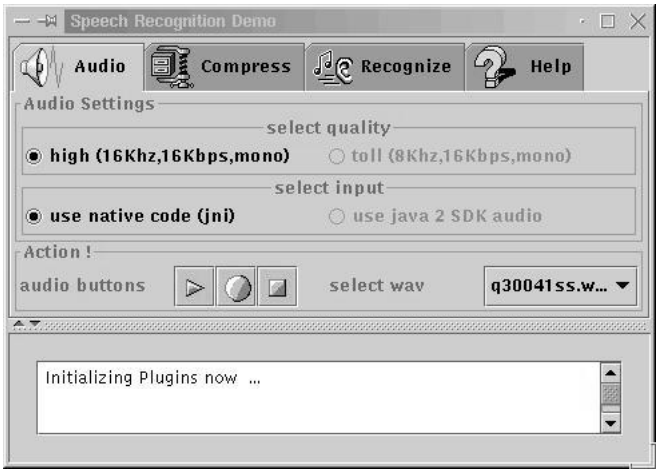
Αυτό το *plugin* είναι υπεύθυνο για την δικτυακή επικοινωνία με τον αναγνωριστή. Παίρνει σαν είσοδο το συμπιεσμένο *feature*, το μεταδίδει στον αναγνωριστή και ανακτά το αποτέλεσμα της αναγνώρισης.

5.5.4 Batch recognition applet

Σε αυτή τη παράγραφο περιγράφεται η χρήση των *plugins* από μια απλή εφαρμογή(*applet*) για *batch* αναγνώριση. Όπως φαίνεται και στις επόμενες εικόνες του *applet*, υπάρχουν 4 *panels*(πάνελς). Τα 3 πρώτα χρησιμοποιούν τα 3 *plugins* της προηγούμενης παραγράφου αντίστοιχα, ενώ το 4ο είναι το σύστημα

[4] Συγκεκριμένα ο *Open Sound System (www.4front-tech.com)*

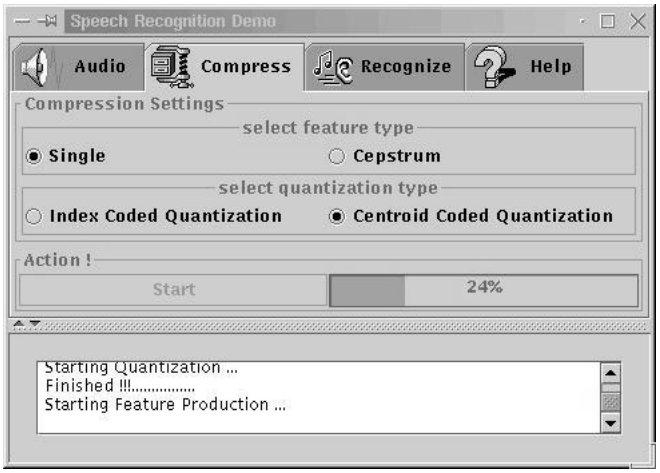
βοήθειας του χρήστη. Σε κάθε ένα από τα 3 πρώτα πάνελς, υπάρχουν διαθέσιμες 2 περιοχές. Η πρώτη (περιοχή επιλογών - *settings area*), επιτρέπει στο χρήστη να καθορίσει τις παραμέτρους λειτουργίας, ενώ η δεύτερη (περιοχή ενεργειών - *action area*), να επιτελέσει την αντίστοιχη λειτουργία του παρόντος πάνελ (π.χ. συμπίεση για το *Compress Panel*). Τέλος, κοινή για όλα τα *panels* είναι η περιοχή μηνυμάτων (*Message Console*), η οποία δίνει πληροφορίες κατά τη διάρκεια της λειτουργίας του πελάτη. Στη συνέχεια εξετάζεται η λειτουργία του κάθε πάνελ χωριστά.



Σχήμα 5.5: AudioPanel

Audio Panel

Το *Audio Panel* (βλέπε Σχήμα 5.5) περιλαμβάνει επιλογές που αφορούν την πιστότητα της ηχογράφησης, καθώς αυτή καθορίζει το *front-end* που θα χρησιμοποιηθεί. Αυτόματα ελέγχεται το είδος *codebooks* που υπάρχουν και αν αυτά έχουν παραχθεί για παράδειγμα, μόνο για υψηλής πιστότητας ήχο, τότε η επιλογή για χαμηλή πιστότητα απενεργοποιείται. Επιπλέον υπάρχει και επιλογή για την χρηση *JNI (java SDK 1)* ή *java SDK 2* κώδικα για την ηχογράφηση. Στην περιοχή ενεργειών, ο χρήστης καλείται να καθορίσει μια πρόταση για είσοδο στο *front-end*. Μπορεί να επιλέξει να ηχογραφήσει μια νέα κυματομορφή χρησιμοποιώντας τα 3 *audio buttons* ή να διαλέξει μια από τις διαθέσιμες από τη μπάρα επιλογής στα δεξιά.



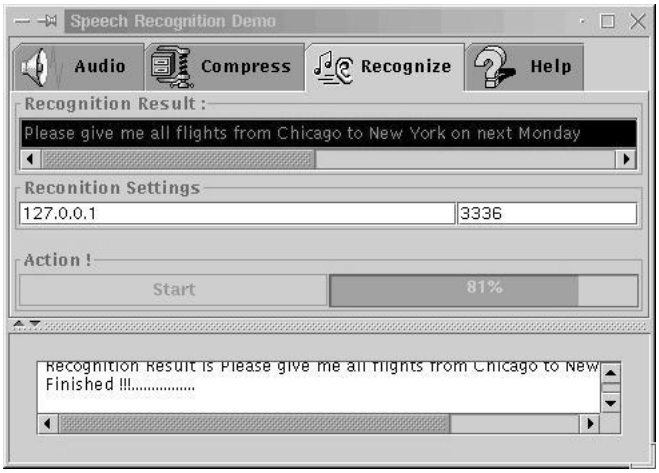
Σχήμα 5.6: Compress Panel

CompressPanel

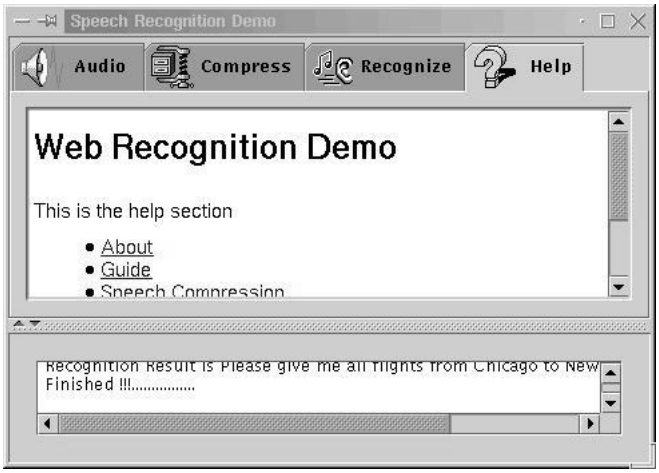
Στο *Compress Panel* (βλέπε Σχήμα 5.6), ο χρήστης επιλέγει το είδος του *feature* που θα παραχθεί (*Single* ή *Cepstrum*) και το είδος κβαντισμού (*Index* ή *Centroid Coded*). Στην περιοχή ενεργειών, υπάρχουν το κουμπί έναρξη (*Start button*) και η μπάρα προόδου (*progress bar*) που δείχνει το ποσοστό της εκτελούμενης εργασίας που έχει γίνει έως τώρα. Σε αυτό το πάνελ επιτελούνται δυο εργασίες : η παραγωγή του *feature* από το *front-end* και ο κβαντισμός με βάση το επιλεγμένο σχήμα κωδικοποίησης.

RecognizePanel

Στο *Recognize Panel*, ο χρήστης απλά επιλέγει το κουμπί *start* για να σταλθεί η κωδικοποιημένη φωνή στον εξυπηρετητή για αναγνώριση. Και εδώ όπως και στη περιοχή ενεργειών του προηγούμενου πάνελ, υπάρχει μπάρα προόδου που ενημερώνει το χρήστη για τη πρόοδο της αναγνώρισης . Στη περιοχή επιλογών, ο χρήστης μπορεί να καθορίσει την διεύθυνση *IP* και το *port* του μηχανήματος στο οποίο τρέχει ο *daemon* (στη περίπτωση που η εφαρμογή δεν είναι *applet*). Στην εικόνα 5.7 βλέπουμε ότι ο *daemon* τρέχει στο τοπικό μηχάνημα, αλλά φυσικά οποιαδήποτε έγκυρη διεύθυνση μπορεί να καθοριστεί (π.χ. 147.27.8.126). Το *Network plugin* θα αναλάβει την επικοινωνία με τον αναγνωριστή και μόλις ανακτηθεί το αποτέλεσμα της αναγνώρισης, θα εμφανιστεί στη μπάρα κειμένου που βρίσκεται στο πάνω μέρος του πάνελ (*Recognition Result*).



Σχήμα 5.7: Recognize Panel



Σχήμα 5.8: Help Panel

HelpPanel

Το τέταρτο και τελευταίο πανελ είναι η βοήθεια για τη λειτουργία του *applet* που περιλαμβάνει :

- Αναλυτικό οδηγό για όλες τις λειτουργίες του *applet*
- Τεχνική ανάλυση που περιλαμβάνει την τεχνολογία συμπίεσης φωνής που χρησιμοποιεί το *applet*

5.6 Συμπεράσματα

- Αρχικά, τονίσαμε ότι η πιο σημαντική εφαρμογή αναγνώρισης φωνής στο χώρο των τηλεπικοινωνιών, είναι η πρόσβαση σε πληροφορίες μέσω δικτύου φωνής (*PSTN* ή *GSM*) από σταθερά ή κινητά τηλέφωνα.

- Στη συνέχεια αναλύσαμε τις ραγδαίες εξελίξεις στο χώρο των ασύρματων επικοινωνιών με την εμφάνιση συσκευών όπως *pdas* και κινητά νέας γενεάς με δυνατότητα πρόσβασης σε ασύρματα δίκτυα δεδομένων (*802.11, GPRS*). Η χρήση της κωδικοποίησης φωνής αυτής της εργασίας σε τέτοιες συσκευές, καθιστά δυνατή την πρόσβαση σε υπηρεσίες αναγνώρισης φωνής με βάση το μοντέλο πελάτη-εξυπηρετητή με ελάχιστο κόστος, λόγω του πολύ μικρού ρυθμού δεδομένων (*2kbps*).

- Τελος, δώσαμε ένα παράδειγμα εφαρμογής κατανεμημένης αναγνώρισης φωνής, στο οποίο χρησιμοποιήσαμε τα συστήματα *Yarrow & Decipher* στον πελάτη και εξυπηρετητή αντίστοιχα. Εξετάσαμε την λειτουργικότητα του πελάτη και τον τρόπο επικοινωνίας με τον αναγνωριστή, δείχνοντας έτσι πως μπορεί να χτιστεί μια πραγματική εφαρμογή/υπηρεσία πάνω σε αυτό το μοντέλο.

Κεφάλαιο 6

Κωδικοποίηση φωνής σε pda σε πραγματικό χρόνο

6.1 Εισαγωγή

Στο κεφάλαιο αυτό θα εξετάσουμε το σενάριο πελάτη-εξυπηρετητή του προηγούμενου κεφαλαίου, όπου για πελάτη χρησιμοποιούμε ένα *pda* και η αναγνώριση γίνεται *live*, όπως θα απαιτούσε οποιαδήποτε πραγματική εφαρμογή. Λέγοντας *live* αναγνώριση, εννοούμε ότι η κωδικοποίηση και αναγνώριση της φωνής θα γίνεται από τη στιγμή που αρχίζει η ηχογράφηση του προς αναγνώριση μηνύματος και όχι μετά το τέλος αυτής όπως συμβαίνει στη περίπτωση της *batch*[1] αναγνώρισης. Φυσικά για να είναι επιτυχής η εφαρμογή θα πρέπει η *live* κωδικοποίηση να γίνεται σε πραγματικό χρόνο (*Real Time*).

6.2 Χαρακτηριστικά του Zaurus pda

Το *pda* που επιλέχθηκε είναι το *Zaurus SL5000D* [2] της *Sharp*, ένα *linux based pda* της κατηγορίας *pocket pc*. Είναι το πρώτο *pda* στο οποίο έχει γίνει διαθέσιμη η *J2ME CDC* και διαθέτει τα παρακάτω χαρακτηριστικά :

- 206 Mhz StrongArm processor, 16 MB ROM, 32 MB storage + RAM
- Linux kernel 2.4.6-rmk1-np2-embedix, J2ME CDC personal profile

[1] Οι δυο μορφές αναγνώρισης (*batch* & *live*) χρησιμοποιούν και διαφορετικά *features* για την αναγνώριση : τα *zero-mean cepstrum* & *rasta cepstrum* αντίστοιχα

[2] *Developer* έκδοση του *pda* : Κερδήθηκε σε διεθνή διαγωνισμό προγραμματισμού της *Trolltech* για *Zaurus* εφαρμογές

Δυστυχώς όπως όλα τα *pdas*, έτσι και το *Zaurus* δε διαθέτει μονάδα κινητής υποδιαστολής *Floating Point Unit (fpu)*. Αν σκεφτεί κανείς ότι η κωδικοποίηση βασίζεται σε *floating-point* αριθμητική, τότε ίσως αναρωτηθεί πως θα γίνεται να τρέχει το *Yarrow* ή οποιαδήποτε άλλη εφαρμογή που χρησιμοποιεί *floating-point* αριθμητική, στο *Zaurus*. Η απάντηση είναι ότι υπάρχει ένα *module* στο *kernel* που προσομοιώνει όλες τις *floating-point* λειτουργίες. Το κόστος όμως της προσομοίωσης είναι αρκετά ακριβό σε ταχύτητα, με αποτέλεσμα η παραγωγή και μόνο του *cepstrum* να απαιτεί χρόνο μεγαλύτερο απο *Real Time* (1.2 φορές). Έτσι αποφασίστηκε να δημιουργηθεί μια *fixed-point* έκδοση του *Yarrow*, με την ελπίδα ότι θα τρέχει σε πραγματικό χρόνο, αφού δεν θα χρησιμοποιεί αριθμητική κινητής υποδιαστολής και έτσι θα είναι πιο γρήγορη. Τα ερωτήματα που προέκυψαν όσο αφορά τη χρήση αριθμητικής σταθερής υποδιαστολής είναι :

1. Θα είναι εφικτή η κωδικοποίηση σε πραγματικό χρόνο;
2. Θα επηρεάσει την ακρίβεια της κωδικοποίησης ώστε να μειώσει την απόδοση της αναγνώρισης;

6.3 Floating & fixed point αριθμητική

Η αριθμητική κινητής υποδιαστολής χρησιμοποιείται σε όλους τους σύγχρονους υπολογιστές για όλες τις εφαρμογές που χρησιμοποιούν πραγματικούς αριθμούς. Μάλιστα τα *fpus* έχουν εξελιχθεί σε τέτοιο βαθμό που οι πράξεις με πραγματικούς αριθμούς έχουν γίνει τόσο γρήγορες όσο και οι πράξεις με ακεραίους. Ο βασικός λόγος που δε χρησιμοποιούνται σε συσκευές όπως τα *pdas* είναι ότι οι περισσότερες τέτοιες συσκευές φτιάχτηκαν για να τρέχουν *integer based* εφαρμογές. Επιπλέον λόγοι αποτελούν το επιπρόσθετο κόστος ενός *fpu* όσο αφορά τη τιμή, την κατανάλωση ενέργειας και τον επιπλέον χώρο.

Η αριθμητική σταθερής υποδιαστολής υπήρξε ιδιαίτερα διαδεδομένη πολλά χρόνια πριν, που πολλοί υπολογιστές δεν διέθεταν καν *fpus* ή όταν αυτά ήταν ακόμα ιδιαίτερα αργά. Σήμερα βρίσκει εφαρμογή σε περιπτώσεις *embedded* συσκευών όπως αυτές των *pdas* και *smartphones*.

6.3.1 Αναπαράσταση floating-point αριθμών

Ένας τρόπος αναπαράστασης των πραγματικών αριθμών είναι η λεγόμενη επιστημονική αναπαράσταση (*scientific notation*), στην οποία ένας αριθμός με ένα μη δεκαδικό ψηφίο (*mantissa*) πολλαπλασιάζεται με το δέκα σε κάποια δύναμη (*exponent*) π.χ. 2.657×10^{-23}. Η *floating-point* αριθμητική ακολουθεί

το *ANSI-IEEE 754* πρότυπο το οποίο χρησιμοποιεί την επιστημονική αναπαράσταση με τη διαφορά ότι σαν βάση χρησιμοποιείται το 2 αντί για το 10 και έχει τη μορφή : $N = (-1)^S \times M \times 2^{E-127}$ (*float number format*).

Τα *bits* 0-22 χρησιμοποιούνται για τη *mantissa* M (με φορματ $M = 1.m_{22}...m0$), τα *bits* 23-30 για το *exponent* E και το *bit* 31 για το πρόσημο S. Χρησιμοποιώντας λοιπόν αυτό το σχήμα, ο μεγαλύτερος αριθμός που μπορεί να αναπαρασταθεί είναι ο $\pm(2 - 2^{-23}) \times 2^{128} = \pm 6.8 \times 10^{38}$, ενώ ο μικρότερος είναι $\pm 1.0 \times 2^{-127} = \pm 5.9 \times 10^{-39}$.

6.3.2 Αναπαράσταση fixed-point αριθμών

Μπορούμε επίσης να αναπαραστήσουμε πραγματικούς αριθμούς με χρήση ακεραίων αν θεωρήσουμε ένα φανταστικό σημείο στο οποίο θα βρίσκεται η υποδιαστολή και κάνοντας όλες τις πράξεις έχοντας αυτό το δεδομένο υπόψη. Η θέση της υποδιαστολής καθορίζεται έμμεσα, από τον αριθμό *bits* που θα διατεθούν για την αναπαράσταση του ακέραιου και δεκαδικού μέρους, που καθορίζουν το εύρος κάλυψης τιμών (*dynamic range*) και την ακρίβεια αναπαράστασης του δεκαδικού μέρους (*precision*) αντίστοιχα.

Για παράδειγμα, αν διατεθούν 16 *bits* τόσο για το ακέραιο όσο και για το δεκαδικό μέρος, θα μπορούμε να αναπαραστήσουμε πραγματικούς αριθμούς μεταξύ -32768.0 και $+32767.99998$ (το διάστημα μεταξύ 2 διαδοχικών αριθμών θα είναι $2^{-16} = 0.000015259$). Αν επιθυμούμε μεγαλύτερο διάστημα κάλυψης τιμών, μπορούμε να διαθέσουμε π.χ. 24 *bits* για το ακέραιο μέρος αλλά τότε μένουν μόνο 8 *bits* για το δεκαδικό μέρος. Σε αυτή τη περίπτωση, θα μπορούμε να αναπαραστήσουμε τιμές από -8388608 έως +8388607.996 αλλά με *precision* μόλις 0.00390625 (2^{-8}).

Το πως θα κατανεμηθούν τελικά τα *bits* εξαρτάται τόσο από το επιθυμητό *precision*, όσο και από το διάστημα κάλυψης τιμών και κατά βάση εξαρτάται από την φύση της εφαρμογής που θέλουμε να μετατρέψουμε σε *fixed-point arithmetic*. Ο αριθμός των *bits* που διατίθονται για την αναπαράσταση του δεκαδικού μέρους (*precision bits*) καθορίζει και το όνομα που δίνουμε στο *fixed-point* αριθμό (ένας *fixed-8* αριθμός π.χ. χρησιμοποιεί 8 *precision bits*).

6.3.3 Σύγκριση floating & fixed point αναπαράστασης

Αν και οι λέξεις *floating* & *fixed* αναφέρονται στην υποδιαστολή, το σωστό θα ήταν να αναφερόταν στα διαστήματα μεταξύ των διαδοχικών τιμών που αναπαριστούν. Αυτό ισχύει, γιατί στη πραγματικότητα με *floating-point* αριθμητική η υποδιαστολή είναι σταθερή (όχι *floating*), σε αντίθεση με τη *fixed-point* αριθμητική, όπου μπορούμε να φανταστούμε την υποδιαστολή να κινείται (όχι

fixed) ανάλογα με το *precision* που επιθυμούμε.

Επιπλέον σε *fixed-point* αριθμητική το διάστημα μεταξύ των διαδοχικών τιμών που μπορούμε να αναπαραστήσουμε (το οποίο όπως είδαμε καθορίζεται από το επιθυμητό *precision*) είναι σταθερό (*fixed*). Αντίθετα, με *floating-point* αριθμητική, τα διαστήματα αυτά έχουν πολύ μικρές τιμές κοντά στο μηδέν οι οποίες γίνονται πολύ μεγάλες καθώς πηγαίνουμε προς τις μέγιστες και ελάχιστες τιμές αντίστοιχα (εξαιρετικά μεγάλη ακρίβεια κοντά στο μηδέν η οποία σταδιακά μειώνεται καθώς η απόλυτη τιμή του αριθμού αυξάνεται).

6.3.4 Fixed-point αριθμητική

Οι βασικές πράξεις με *fixed-point* αριθμούς είναι φυσικά παρόμοιες με αυτές των ακεραίων, με τη διαφορά βέβαια ότι θα πρέπει πάντα να παίρνουμε υπόψη μας το πως επηρεάζει το δεκαδικό μέρος την πράξη. Οι πράξεις μετατροπής από και προς *fixed-8* αριθμούς, καθώς και οι πράξεις μεταξύ *fixed-8* αριθμών ακολουθούν στη συνέχεια :

```
itofx(x) (x << 8)           // Integer to fixed point
ftofx(x) (x * 256)          // Float to fixed point
fxtoi(x) (x >> 8)           // Fixed point to integer
fxtof(x) ((float)(x) / 256) // Fixed point to float

Addfx(x,y) (x + y)          // Add 2 fixed numbers
Subfx(x,y) (x - y)          // Subtract 2 fixed numbers
Mulfx(x,y) ((y * x) >> 8)   // Multiply 2 fixed numbers
Divfx(x,y) ((y << 8) / x)   // Divide 2 fixed numbers
```

Ας αρχίσουμε με τη μετατροπή ακεραίων σε *fixed-8* αριθμούς. Το μόνο που χρειάζεται να γίνει, είναι απλά να δημιουργήσουμε το δεκαδικό μέρος κάνοντας αριστερή ολίσθηση (*shift*) του ακεραίου κατά 8 *bits*, που αντιστοιχεί σε πολλαπλασιασμό με 256. Στην περίπτωση μετατροπής από *floats*, επίσης απλά πολλαπλασιάζουμε με 256 (αν κάναμε ολίσθηση 8 *bits* θα πέρναμε λάθος αποτέλεσμα όμως, καθώς όπως είδαμε παραπάνω το φορμάτ των *floats* είναι διαφορετικό από αυτό των ακεραίων). Το ακριβώς αντίθετο ισχύει όταν θέλουμε να μετατρέψουμε ένα *fixed-8* αριθμό σε ακέραιο ή *float*. Στην πρώτη περίπτωση κάνουμε ολίσθηση δεξιά κατά 8 *bits* (ουσιαστικά εξαλείφοντας το δεκαδικό μέρος), ενώ στη δεύτερη απλά διαιρούμε με 256.

Ας εξετάσουμε τώρα τις πράξεις μεταξύ *fixed-8* αριθμών. Για την πρόσθεση και την αφαίρεση δεν χρειάζεται να κάνουμε απολύτως τίποτα : είναι ακριβώς σαν την πρόσθεση και αφαίρεση ακεραίων. Αυτό προκύπτει εύκολα αν σκεφτούμε ότι ένας *fixed-8* αριθμός προκύπτει από ένα ακέραιο (ή

float) απλά πολλαπλασιάζοντας με τον αριθμό $f = 2^8$. Τότε αν x και y είναι δυο π.χ. ακέραιοι, η *fixed-8* αναπαράσταση του αθροίσματός τους θα είναι : $(x + y) \times f = x \times f + y \times f$, δηλαδή απλά το άθροισμα των *fixed-8* αναπαραστάσεών τους. Με την ίδια λογική, η *fixed-8* αναπαράσταση του γινομένου τους θα είναι : $(x \times y) \times f = (x \times f) \times (y \times f)/f$, δηλαδή το γινόμενο των *fixed-8* αναπαραστάσεών τους δια τον αριθμό f. (δεξιά ολίσθηση κατά 8). Για τη διαίρεση προκύπτει ανάλογα, ότι χρειάζεται να πολλαπλασιάσουμε με f (αριστερή ολίσθηση κατά 8).

6.3.5 Θέματα πράξεων με fixed-point αριθμούς

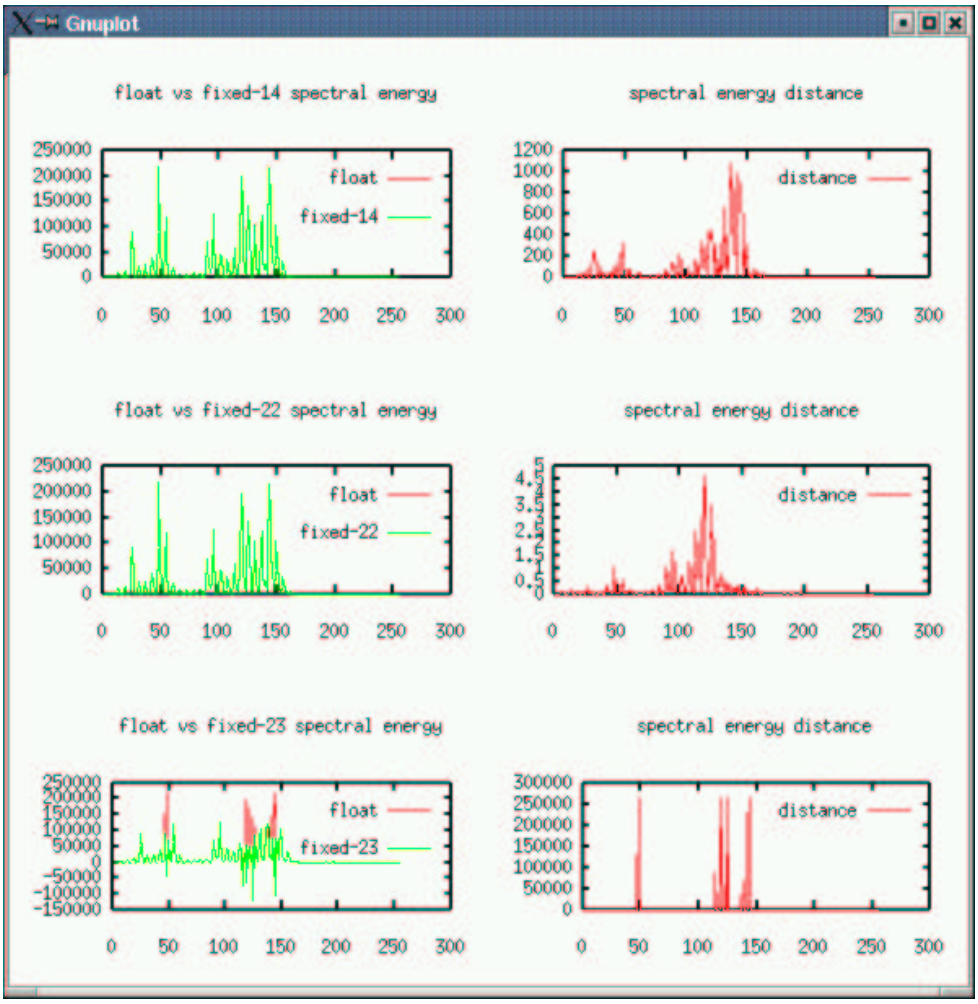
Ένα σημείο που πρέπει να τονισθεί εδώ είναι ότι στις πράξεις με *fixed-point* αριθμούς είναι εύκολο να χαθεί η ακρίβεια ή να προκύψουν πολύ λανθασμένα αποτελέσματα. Το πρώτο συμβαίνει όταν το *precision* που χρησιμοποιούμε είναι πολύ μικρό σε σχέση με το απαιτούμενο (π.χ πράξεις με πολύ μικρούς αριθμούς) ή όταν οι υπολογισμοί απαιτούν πολύ μεγάλο αριθμό πράξεων, οπότε σταδιακά η ακρίβεια μειώνεται.

Το πιο δύσκολο πρόβλημα όμως, είναι η αποφυγή *overflows* που είναι και εύκολο να συμβούν ακόμα και στη περίπτωση πράξεων μεταξύ όχι και τόσο μεγάλων αριθμών, αφού ένα μέρος του ακεραίου (π.χ. 16 από τα 32 *bits*) χρειάζεται για την αναπαράσταση του δεκαδικού μέρους. Έτσι π.χ. ακόμα και μια πρόσθεση μεταξύ 20000.0 και 30000.0 δίνει -15536.0 αν δεν αποθηκευτεί προηγουμένος το αποτέλεσμα σε ένα τύπο ακεραίου με περισσότερα *bits* (*long* που έχει 8 *bytes* αντί για 4 που έχει ο *integer*). Το ίδιο πρόβλημα είναι βέβαια ακόμα πιο έντονο στη περίπτωση του πολλαπλασιασμού, όπου είναι πολύ εύκολο να συμβεί κάποιο *overflow* π.χ. πολλαπλασιασμός 500.0 και 100.0.

Αν σκεφτεί κανείς ότι τα δυο αυτά προβλήματα προκύπτουν από την ταυτόχρονη ανάγκη να διατηρήσουμε μεγάλο *dynamic range* από τη μια και επίσης μεγάλη ακρίβεια από την άλλη, τότε γίνεται φανερό ότι πρέπει να βρεθεί η λύση που θα δίνει τα καλύτερα αποτελέσματα για τη συγκεκριμένη περίπτωση, όπως φαίνεται στον παρακάτω πίνακα.

Στον Πίνακα 6.1, συγκρίνουμε την απόσταση *fixed-point* εκδόσεων για διαφορετικό *precision*, από τη *float* έκδοση της φασματικής ενέργειας (*Spectral Energy*) ενός συγκεκριμένου *frame* φωνής. Φαίνεται καθαρά, ότι τα *overflows* στην περίπτωση των 23 *bits* προκαλούν πολύ μεγαλύτερη απόσταση από ότι ο θόρυβος λόγω του μικρότερου *precision* (βλέπε επίσης Σχήμα 6.1). Ωστόσο το πρόβλημα των *overflows* μπορεί να αποφευχθεί στη γενική περίπτωση, απλά χρησιμοποιώντας λιγότερα (π.χ. 22) *precision bits* ή αλλάζοντας την πράξη του πολλαπλασιασμού (π.χ. σε $(x >> 1) \times (y >> 1) >> (23 - 2)$).

Με αυτό τον τρόπο αναμένεται ο αριθμός των *overflows* να περιοριστεί σε



Σχήμα 6.1: **Σύγκριση spectral energy ενός frame φωνής μεταξύ float & fixed-point εκδόσεων.** Τα 23 *bits* εξασφαλίζουν σχεδόν μηδενικές διαφορές για την πλειονότητα των σημείων, προκαλούν όμως *overflow*. Αντίθετα με 14 *bits* υπάρχει εμφανής διαφορά για όλα σχεδόν τα σημεία. Τα 22 *bits* εξασφαλίζουν τη μικρότερη απόσταση από τη float έκδοση για το συγκεκριμένο *frame*.

precision bits	Spectral Energy distance
12	44195
14	17775
16	6713
18	767
20	317
22	53
23	1521030

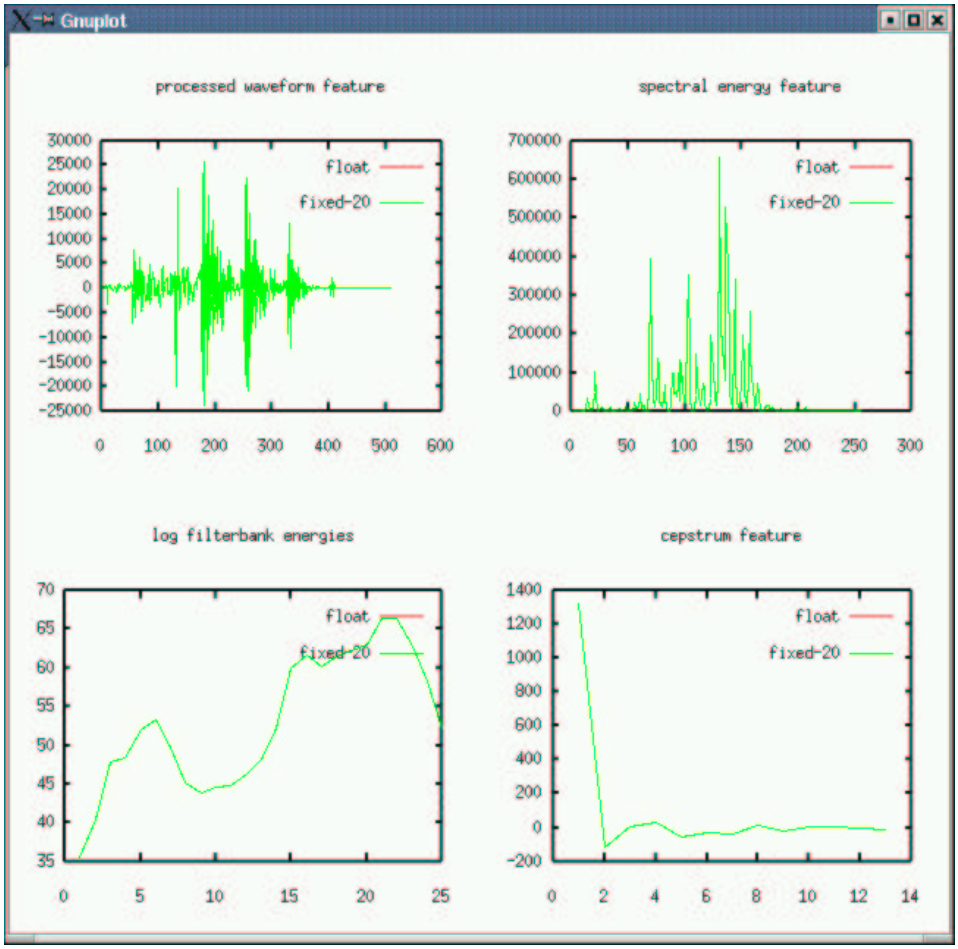
Πίνακας 6.1: **Επίδραση αριθμού precision bits στο Spectral Energy distance με τη float έκδοση**. Τα 22 *bits* δίνουν την ελάχιστη απόσταση μεταξύ *fixed* & *floating-point* εκδόσεων. Μειώνοντας τον αριθμό των *bits* από 22 σε 12 η απόσταση σταδιακά αυξάνει λόγω του μικρού precision. Αντίθετα, από τα 22 στα 23 *bits* προκύπτει πολύ απότομη και μεγάλη διαφορά, που οφείλεται στην δημιουργία *overflows*, αφού μόνο 18 *bits* (64-2*23) διατίθενται για την αναπαράσταση του ακέραιου μέρους κατά τον πολλαπλασιασμό.

ελάχιστα σημεία ενός πολύ μικρού αριθμού frames, έτσι ώστε ουσιαστικά να έχει ελάχιστη επίδραση στην αναγνώριση (κάτι που θα δούμε και στη συνέχεια). Αντίθετα όμως, ο θόρυβος που θα δημιουργείται από τον μικρό αριθμό *precision bits*, θα είναι εμφανής στα περισσότερα σημεία όλων των *frames* κάτι που τελικά μπορεί να επηρεάσει αρκετά την αναγνώριση όταν ο θόρυβος γίνει σχετικά μεγάλος (πολύ μικρός αριθμός *precision bits*).

6.4 Fixed-point έκδοση του Yarrow

Όπως είδαμε η χρήση *floats* εξασφαλίζει τη δυνατότητα αναπαράστασης πολύ μικρών και πολύ μεγάλων αριθμών. Με τη χρήση ακεραίων (*fixed-point*), κάτι τέτοιο δεν μπορεί να εξασφαλιστεί, γι' αυτό επιβάλεται η προσεκτική εφαρμογή των πράξεων ώστε να αποφευχθούν τα όποια προβλήματα *overflows* & *underflows* ή μείωσης της ακρίβειας. Θεωρητικά, αυτό σημαίνει ότι πρέπει κανείς να γνωρίζει το εύρος τιμών που παίρνει κάθε μεταβλητή, κάτι το οποίο γενικά είναι πολύ δύσκολο.

Αυτό που μπορεί να γίνει στη πράξη όμως, είναι να βρεθούν τα όποια σημεία προκαλούν τέτοια προβλήματα και να γραφτούν πιο προσεκτικά. Για παράδειγμα, ας φανταστούμε ότι έχουμε την έκφραση : $x^2 \times y^2$, όπου το x είναι ένας πάρα πολύ μικρός αριθμός, ενώ το y ένας σχετικά μεγάλος. Τότε είναι πολύ καλύτερα να το αντικαταστήσουμε με την έκφραση : $(x \times y)^2$, γιατί από τη μια δεν θα έχουμε τη μείωση της ακρίβειας του πολλαπλασιασμού δυο



Σχήμα 6.2: **Σύγκριση floating & fixed-point(20 bits) front-end features.** Το πρώτο σχήμα δείχνει την επεξεργασμένη κυματομορφή ακριβώς πριν τον υπολογισμό του *fft*. Τα επόμενα σχήματα συγκρίνουν τη *spectral energy* (αμέσως μετά τον *fft*), την έξοδο των *filterbanks* και το *cepstrum* που προκύπτουν με τις *floating* & *fixed-point* εκδόσεις αντίστοιχα. Όπως φαίνεται, η χρήση *fixed-point* με 20 *bits precision* μπορεί να εγγυηθεί τιμές πολύ κοντινές σε αυτές που προκύπτουν με *floating-point*. Οι διαφορές υπάρχουν, αλλά είναι εξαιρετικά μικρές, τόσο ώστε πρακτικά να μην επηρεάζουν την ακρίβεια κβαντισμού και κατά συνέπεια την ακρίβεια αναγνώρισης.

πολύ μικρών αριθμών και από την άλλη αναιρούμε την περίπτωση να πάρουμε *overflow* πολλαπλασιάζοντας τους δυο μεγάλους αριθμούς.

6.4.1 Ακρίβεια αναγνώρισης fixed-point έκδοσης

Αφού λοιπόν δημιουργήθηκε η *fixed-point* έκδοση του *Yarrow* σύμφωνα με τα παραπάνω, έπρεπε να διαπιστωθεί πόσο καλή ήταν η ακρίβεια αναγνώρισης. Για το σκοπό αυτό έγινε σύγκριση των *features* του *front-end* μεταξύ των *floating-point* & *fixed-point* εκδόσεων (βλέπε Σχήμα 6.2) και προέκυψε πειραματικά (βλέπε Πίνακα 6.2) ότι με 20 *bits* παίρνουμε τα καλύτερα αποτελέσματα.

Στον Πινακα 6.2 φαίνονται τα αποτελέσματα της αναγνώρισης 230 προτάσεων από το *test set* του *ATIS corpus* για διαφορετικό αριθμό *precision bits* της *fixed-point* έκδοσης. Με 20 *precision bits* εξασφαλίζουμε την ίδια επίδοση αναγνώρισης με αυτή της *floating-point* έκδοσης (*WER* 6.78%).

precision bits	12	14	16	18	19	20	21	22	23
WER(%)	22.61	6.89	6.88	6.88	6.78	6.78	6.78	7.11	7.22

Πίνακας 6.2: **Επίδραση αριθμού precision bits στην αναγνώριση**. Για 19-21 *bits* το WER είναι ίδιο με τη float έκδοση. Με 22 και 23 *bits* υπάρχει μια μικρή αύξηση του WER (προφανώς υπάρχουν λίγα μόνο *overflows*). Με 18-14 *bits* δεν υπάρχει ουσιαστική αύξηση του WER σε αντίθεση με τα 12 *bits* που αυξάνεται απότομα.

6.4.2 Ταχύτητα fixed-point έκδοσης

Άρα αφού απαντήθηκε το πρώτο ερώτημα της ακρίβειας της *fixed-point* έκδοσης, έπρεπε να βεβαιωθούμε πως ο υπολογισμός γίνεται και σε πραγματικό χρόνο. Πράγματι, η παραγωγή του *cepstrum* γίνεται σε χρόνο περίπου $0.8 \times RealTime$, ενώ η *floating-point* έκδοση απαιτούσε σχεδόν $1.2 \times RealTime$.

6.5 Fixed-point live έκδοση του Yarrow

Φυσικά μια εφαρμογή *live* κωδικοποίησης φωνής, απαιτεί επιπλέον χρόνο για λειτουργίες όπως :

- Διάβασμα από τον *audio buffer* των δειγμάτων της φωνής
- Τον κβαντισμό του *cepstrum* σε $2Kbps$
- Την αποστολή των δεδομένων στον αναγνωριστή.

Για τη λειτουργία της κωδικοποίησης σε *live mode* απαιτείται να γίνονται συγχρόνως η ηχογράφηση με την επεξεργασία της φωνής. Σε αντίθεση με την κωδικοποίηση σε *batch mode* που η επεξεργασία της φωνής αρχίζει αφού έχουν γίνει πλέον διαθέσιμα όλα τα δείγματα της φωνής, σε *live mode* είναι απαραίτητο να διαβάζονται ανα μικρά χρονικά διαστήματα τα δείγματα της φωνής τα οποία στη συνέχεια θα επεξεργάζονται. Φυσικά δυο απαραίτητες απαιτήσεις είναι να μην χάνεται κανένα δείγμα φωνής κατά την ηχογράφηση αλλά και να υπολογίζονται όλα τα *frames* στη περίπτωση που η κωδικοποίηση δεν γίνεται σε πραγματικό χρόνο.

Για να εξασφαλιστεί η πρώτη απαίτηση, η λύση είναι να διαβάζονται τα δείγματα της φωνής με τον ίδιο ρυθμό με τον οποίο παράγονται. Επειδή ο ρυθμός που παράγονται τα δείγματα είναι γνωστός (αφού εξαρτάται από τις παραμέτρους της δειγματοληψίας - έστω r), τότε αρκεί να διαβάζονται $n = r \times t$ δείγματα κάθε χρονικό διάστημα t. Αυτό μπορεί να επιτευχθεί με τη χρήση ενός *timer* ο οποίος θα ενημερώνει κάθε t *secs* την εφαρμογή ότι πρέπει να διακόψει για να διαβάσει τα n δείγματα. Αυτό όχι μόνο εξασφαλίζει ότι δεν θα χαθούν δείγματα, αλλά επίσης ότι θα μείνει ο μέγιστος δυνατός χρόνος για την κωδικοποίηση της φωνής, αφού ο χρόνος του διαβάσματος των δειγμάτων της φωνής θα είναι μηδαμινός (το διάβασμα από το *audio device* είναι *blocking* - γνωρίζοντας όμως κάθε πότε θα διαβάζονται πόσα δείγματα, αυτός μπορεί να γίνει πολύ μικρός).

Επειδή είναι πιθανό η κωδικοποίηση να μη γίνεται σε πραγματικό χρόνο (π.χ. σε *pdas* με πολύ αργό επεξεργαστή) είναι απαραίτητο να εξασφαλιστεί ότι θα παραχθούν όλα τα *frames*. Για να γίνει αυτό αρκεί όταν διαβαστούν τα δείγματα να αποθηκευτούν και να επεξεργαστούν μόνο όταν τελειώσει η επεξεργασία των προηγούμενων δειγμάτων. Στην αντίθετη περίπτωση που η κωδικοποίηση γίνεται γρηγορότερα από πραγματικό χρόνο, θα πρέπει η εφαρμογή απλά να περιμένει μέχρι να διαβαστεί το επόμενο μπλοκ από δείγματα.

6.5.1 Live recording

Οι ηχογραφήσεις γίνονται για υψηλής πιστότητας φωνή (*1 channel, 16000 samples/sec, 2 bytes/sample*). Ο *Open Sound System (OSS) audio driver* που χρησιμοποιείται έχει υλοποιηθεί στις περισσότερες πλατφόρμες *Unix* και είναι ο *driver* που χρησιμοποιεί και το *Zaurus* (δυστυχώς στη περίπτωση του *Zaurus* ο οδηγός είναι μερικώς υλοποιημένος και επίσης παρουσιάζει αρκετά προβλήματα).

Ο *driver* δεσμεύει ένα *audio buffer* όπου αποθηκεύονται τα ηχογραφημένα δείγματα και αποτελείται από ένα συγκεκριμένο αριθμό μικρότερων *buffers* που ονομάζονται *fragments*. Για τις ρυθμίσεις που θέσαμε ο *driver* αυτόματα

καθορίζει το μέγεθος του *audio buffer* που θα χρησιμοποιήσει, που στη περίπτωσή μας αποτελείται από 256 *fragments* μεγέθους 256 *bytes* ο καθένας (συνολικό μέγεθος $64KB$ ή 65536 *bytes*). Αυτό σημαίνει ότι ο *buffer* μπορεί να κρατήσει δείγματα περίπου 2 δευτερολέπτων (για την ακρίβεια $2.048 = 65536/32000$) και έτσι μια εφαρμογή πρέπει να διαβάζει τα δείγματα το αργότερο κάθε 2.048 δευτερόλεπτα. Από την άλλη η μικρότερη περίοδος που μπορεί να διαβάζει δείγματα είναι ο χρόνος που απαιτείται για το γράψιμο δειγμάτων από το *audio hardware* σε ένα *fragment* του *buffer*, δηλαδή 8 *ms* (2048/256).

Εξαρτάται από την εφαρμογή κάθε πότε θα διαβάζει τα δείγματα. Για κάποιες εφαρμογές που απαιτούν επεξεργασία σε πραγματικό χρόνο, θα μπορούσε το διάστημα αυτό να είναι τα 8 *ms*. Στη περίπτωσή μας το διάβασμα των δειγμάτων γίνεται κάθε 32 *fragments* (256 *ms*) γιατί η υλοποίηση του *driver* στο *Zaurus* δεν επέτρεπε μικρότερες τιμές. Στα 32 *fragments* (μέγεθος 8192 *bytes*) θα αναφερόμαστε απο εδώ και πέρα ως *audio block* ή απλά *block*(μονάδα διαβάσματος από *audio buffer*).

Ο κώδικας που έχει να κάνει με την πρόσβαση στον *driver* είναι υλοποιημένος σε *C*. Η βιβλιοθήκη που προκύπτει γίνεται *cross compile* σε *Strong ARM* αρχιτεκτονική. Ο *java* κώδικας χρησιμοποιεί τη βιβλιοθήκη μέσω του *Java Native Interface - jni*. Στο διάβασμα των δειγμάτων του *block* γίνεται και η μετατροπή των τιμών από *C short* σε *Java short* και στη συνέχεια σε *fixed-point long*.

6.5.2 Περιγραφή live κωδικοποίησης

Η εφαμογή μας λοιπόν θα πρέπει κάθε 256 *ms* να διαβάζει ένα *block* από το *buffer* και μέσα στα 256 *ms* που θα μεσολαβήσουν μέχρι το διάβασμα του επόμενου *block* να έχει υπολογίσει τα *frames* που αντιστοιχούν στα δείγματα του *block*. Λέγοντας υπολογίσει τα *frames*, εννοούμε να έχει υπολογίσει το *cepstrum* να κάνει το κβαντισμό σε $2Kbps$ και να στείλει τα αποτελέσματα στον αναγνωριστή.

Παρακάτω περιγράφεται η *live* λειτουργία της κωδικοποίησης (η έξοδος είναι από ένα *Linux* υπολογιστή $PIII - 800$).

```
AUDIO SETTINGS :
 number of bits     : 16
 number of channels : 1
 rate               : 16000

BUFFER PARAMETERS :
Total # of fragments allocated                    : 256
```

```
Size of fragments                                        : 256
# of full fragments that can be read without blocking  : 0
# of bytes that can be read without blocking           : 0

BUFFER STATUS : available fragments 0, available bytes 0
took 254061 microsecs to read 8192 bytes
BUFFER STATUS : available fragments 0, available bytes 4

CREATING TIMER THREAD (256ms delay, 256ms period)
-->frame 0 completed in 15 ms
-->frame 1 completed in 8 ms
...
-->frame 22 completed in 1 ms

BUFFER STATUS : available fragments 31, available bytes 8140
took 9610 microsecs to read 8192 bytes
BUFFER STATUS : available fragments 1, available bytes 256
WAITED 193 ms to read audio for index 3680
-->frame 23 completed in 194 ms
-->frame 24 completed in 1 ms
...
-->frame 47 completed in 1 ms

BUFFER STATUS : available fragments 33, available bytes 8590
took 93 microsecs to read 8192 bytes
BUFFER STATUS : available fragments 1, available bytes 402
WAITED 225 ms to read audio for index 7680
-->frame 48 completed in 226 ms
...
-->frame 73 completed in 1 ms

BUFFER STATUS : available fragments 32, available bytes 8396
took 90 microsecs to read 8192 bytes
BUFFER STATUS : available fragments 0, available bytes 208
WAITED 220 ms to read audio for index 11840
-->frame 74 completed in 221 ms
-->frame 75 completed in 1 ms
...
-->frame 99 completed in 1 ms
```

Αρχικά η εφαρμογή ανοίγει το *audio device* και τυπώνει τις παραμέτρους δειγματοληψίας (βλέπε AUDIO SETTINGS) και του *audio buffer*(βλέπε BUFFER PARAMETERS). Για κάθε διάβασμα του *block*, τυπώνονται ο αριθμός των διαθέσιμων *fragments* & *bytes* του *audio buffer*, ακριβώς πριν και μετά το διάβασμα, καθώς και ο χρόνος που χρειάστηκε για το διάβασμα του *block* (βλέπε *BUFFER STATUS*). Αυτό μας επιτρέπει να εξετάσουμε με κάθε λεπτομέρεια την λειτουργία της εφαρμογής.

Κατά το διάβασμα του πρώτου *block*, δεν υπάρχουν δείγματα στο *buffer*, για αυτό η εφαρμογή θα πρέπει να μπλοκάρει για περίπου 254 *ms* για να μπορέσει να τα διαβάσει. Αμέσως μετά δημιουργείται ο *timer*(βλέπε CREATING TIMER THREAD), ο οποίος θα αναλαμβάνει κάθε 256 *ms* να ενημερώνει την εφαρμογή να διαβάσει το επόμενο *block*, αρχίζοντας τη λειτουργία του μετά από 256 *ms*, στα οποία θα πρέπει να επεξεργαστούν τα δείγματα του πρώτου *block*.

Στην επεξεργασία των δειγμάτων του πρώτου *block* υπολογίζονται τα *frames* 0-22, αλλά για να υπολογιστεί και το 23ο *frame* χρειάζονται τα δείγματα 3680-4192. Επειδή ένα *block* περιέχει 4096 δείγματα (8192 *bytes*), η εφαρμογή θα πρέπει να περιμένει περίπου 193*ms*(βλέπε WAITED 193), μέχρι να διαβάσει το δεύτερο *block*, που περιέχει τα δείγματα 4096 έως 8192. Το δεύτερο *block* διαβάζεται σε 9610 μ*s*, γιατί υπάρχουν διαθέσιμα 8140 από τα 8192 *bytes*.

Σε αυτές τις περιπτώσεις που για να υπολογισθεί ένα *frame* χρειάζονται δείγματα από δυο διαδοχικά *blocks*, ο χρόνος υπολογισμού του *frame* θα εμπεριέχει και το χρόνο που θα χρειαστεί μέχρι να διαβαστεί το δεύτερο *block*. Για αυτό ο χρόνος υπολογισμού του *frame* 23 φαίνεται να είναι 194 *ms* (193 *ms* η αναμονή διαβάσματος του επόμενου μπλοκ συν 1 *ms* για τον πραγματικό υπολογισμό του).

Στη συνέχεια υπολογίζονται τα *frames* 23-47 και μένουν 225 *ms* μέχρι να διαβαστεί το επόμενο *block*, το οποίο διαβάζεται σε μόλις 93 μ*s*, αφού υπάρχουν 8590 διαθέσιμα *bytes* στον *audio buffer*. Στη συνέχεια υπολογίζονται τα *frames* 48-73, μένουν 220 *ms* μέχρι να διαβαστεί το επόμενο *block*, κ.ο.κ.

6.5.3 Live κωδικοποίηση στο Zaurus

Στη συνέχεια ακολουθούν οι μετρήσεις για τη περίπτωση του *Zaurus*. Όπως αναφέρθηκε και παραπάνω, ο οδηγός είναι μερικώς υλοποιημένος για το *Zaurus* και επίσης παρουσιάζει άλλα προβλήματα ως προς τις αποδεκτές παραμέτρους δειγματοληψίας, τα επιτρεπόμενα μεγέθη του *block* για το διάβασμα των δειγμάτων κ.α. Έπίσης, δεν υπάρχει η δυνατότητα να ελέγξουμε την κατάσταση του *audio buffer* πριν διαβάσουμε κάποιο *block* (όπως είδαμε στη προηγούμενη έξοδο).

Όπως φαίνεται στις μετρήσεις (όπου δεν τυπώνονται οι χρόνοι υπολογισμού των *frames* για να δειχθεί η συμπεριφορά της εφαρμογής στο *long run*), τα 4 πρώτα *blocks*, δηλαδή περίπου το πρώτο δευτερόλεπτο, τα *frames* δεν υπολογίζονται σε πραγματικό χρόνο (δεν φαίνεται να περιμένει η εφαρμογή μεταξύ του διαβάσματος των πρώτων *blocks*). Φυσικά, στις περιπτώσεις που ο υπολογισμός δε γίνεται σε πραγματικό χρόνο, τα δείγματα του νέου *block* που διαβάστηκαν, θα επεξεργαστούν μόνο αφότου τελειώσει ο υπολογισμός των *frames* του προηγούμενου *block*.

Μετά το διάβασμα του τέταρτου *block* όμως, ο υπολογισμός επιτυγχάνεται σε πραγματικό χρόνο και μένουν και κάποια *ms* που χρειάζεται να περιμένει η εφαρμογή μέχρι να διαβαστεί το επόμενο *block*. Άρα τρέχει σε πραγματικό χρόνο μετά το πρώτο δευτερόλεπτο, πράγμα που πρακτικά δεν αποτελεί πρόβλημα, αφού υποθέτουμε ότι καμιά πρόταση που θα θέλαμε να αναγνωρίσουμε δεν θα έχει διάρκεια μικρότερη από ένα δευτερόλεπτο.

```
took 385653 microsecs to read 8192 bytes
CREATING TIMER THREAD (256ms delay, 256ms period)
took 564 microsecs to read 8192 bytes
took 413 microsecs to read 8192 bytes
took 411 microsecs to read 8192 bytes
took 413 microsecs to read 8192 bytes
WAITED 15 ms to read audio for index 16000
took 414 microsecs to read 8192 bytes
WAITED 36 ms to read audio for index 20000
took 412 microsecs to read 8192 bytes
WAITED 38 ms to read audio for index 24160
took 414 microsecs to read 8192 bytes
WAITED 41 ms to read audio for index 28160
took 412 microsecs to read 8192 bytes
WAITED 34 ms to read audio for index 32320
took 411 microsecs to read 8192 bytes
WAITED 38 ms to read audio for index 36480
took 412 microsecs to read 8192 bytes
WAITED 43 ms to read audio for index 40480
took 410 microsecs to read 8192 bytes
WAITED 32 ms to read audio for index 44640
took 414 microsecs to read 8192 bytes
...
```

Στη παρακάτω έξοδο, φαίνεται αναλυτικά η καθυστέρηση που εισαγάγει η επεξεργασία του πρώτου *block*. Μόνο 13 *frames* υπολογίζονται από τα 23 που

θα έπρεπε να υπολογιστούν. Η καθυστέρηση αυτή οφείλεται καθαρά στο γεγονός ότι χρειάζεται κάποιος χρόνος για να αρχίσει η ιδεατή μηχανή της *java* να κάνει χρήση του *jit compiler*, ώστε να επιταχύνει τις μεθόδους που καλούνται πιο συχνά. Έτσι, ο χρόνος υπολογισμού του *frame* είναι 69 *ms* για το πρώτο *frame*, αλλά σταδιακά μειώνεται σε 8-9 *ms* για όλα τα υπόλοιπα *frames* και έτσι πολύ γρήγορα, μετά το πρώτο δευτερόλεπτο, όπως είδαμε προηγουμένως, ο υπολογισμός γίνεται τελικά εντός πραγματικού χρόνου (8-9 *ms* για τον υπολογισμό κάθε *frame* από τα 10 που αντιστοιχούν στον υπολογισμό σε πραγματικό χρόνο του κάθε *frame*).

```
took 385641 microsecs to read 8192 bytes
CREATING TIMER THREAD (256ms delay, 256ms period)
-->frame 0 completed in 69 ms
-->frame 1 completed in 47 ms
-->frame 2 completed in 15 ms
-->frame 3 completed in 22 ms
-->frame 4 completed in 10 ms
-->frame 5 completed in 19 ms
-->frame 6 completed in 11 ms
-->frame 7 completed in 12 ms
-->frame 8 completed in 9 ms
-->frame 9 completed in 9 ms
-->frame 10 completed in 9 ms
-->frame 11 completed in 8 ms
-->frame 12 completed in 9 ms

took 563 microsecs to read 8192 bytes
-->frame 13 completed in 26 ms
-->frame 14 completed in 9 ms
-->frame 15 completed in 9 ms
...
```

6.6 Συμπεράσματα

Σε αυτό το κεφάλαιο αναφερθήκαμε στην διαδικασία που ακολουθήσαμε προκειμένου να καταστήσουμε τη *live* κωδικοποίηση φωνής στο *Zaurus* εφικτή σε πραγματικό χρόνο, ώστε να μπορεί να χρησιμοποιηθεί σε *live* αναγνώριση. Για να γίνει αυτό χρειάστηκε να μετατραπεί το *Yarrow* από ένα *floating-point* & *batch* σύστημα σε ένα *fixed-point* & *live* σύστημα.

- Το *Zaurus* δεν διαθέτει μονάδα κινητής υποδιαστολής, με αποτέλεσμα η κωδικοποίηση της φωνής να μη μπορεί να γίνει σε πραγματικό χρόνο. (Ο χρόνος υπολογισμού του *cepstrum* είναι $1.2 \times RealTime$). Για αυτό το λόγο κρίθηκε απαραίτητο να επεκτείνουμε το *Yarrow* ώστε να χρησιμοποιεί *fixed-point* αριθμητική.

- Η *fixed-point* έκδοση με 20 *bits* αποδείχτηκε ότι προσεγγίζει πολύ καλά την *floating-point* έκδοση, αφού τα 20 *bits* εξασφαλίζουν μικρή απώλεια ακρίβειας από τη μια και από την άλλη αποφυγή όποιων *overflows* και πειραματικά προέκυψε ότι έχει την ίδια επίδοση αναγνώρισης με την *floating-point* έκδοση. Επίσης είναι πιο γρήγορη από τη *floating-point* έκδοση (ο χρόνος υπολογισμού του *cepstrum* είναι $0.8 \times RealTime$).

- Στη *live* έκδοση, απαιτείται να γίνεται η κωδικοποίηση της φωνής στο διάστημα μεταξύ του διαβάσματος των δειγμάτων της φωνής από το *audio device*. Εξασφαλίστηκε ότι δεν χάνονται δείγματα φωνής, αφού αυτά διαβάζονται με τον ίδιο ρυθμό με τον οποίο παράγονται και έτσι μένει και ο μέγιστος δυνατός χρόνος για την κωδικοποίηση της φωνής. Επιπλέον, παράγονται σωστά όλα τα *frames*, άσχετα με το αν η κωδικοποίηση γίνεται σε πραγματικό χρόνο ή όχι και έτσι μπορεί να χρησιμοποιηθεί και σε άλλα αργά *pdas*.

- Η *fixed-point live* έκδοση, μπορεί να χρησιμοποιηθεί σε *live* αναγνώριση, με τη κωδικοποίηση να γίνεται σε πραγματικό χρόνο μετά το πρώτο δευτερόλεπτο, κάτι που είναι απολύτως αποδεκτό, αφού στη πράξη καμιά πρόταση που θα θέλαμε να ανγνωρίσουμε δεν θα έχει διάρκεια μικρότερη από ένα δευτερόλεπτο.

Συμπεράσματα και μελλοντικές επεκτάσεις

Τα κύρια *contributions* αυτής της εργασίας, είναι η εύρεση του πολύ μικρού ρυθμού δεδομένων της τάξης των $2kbps$ με τον οποίο μπορούμε να μεταδώσουμε την ακουστική πληροφορία που είναι απαραίτητη για αναγνώριση και η σε πραγματικό χρόνο κωδικοποίηση φωνής σε συσκευές όπως *pdas*. Με αυτό τον τρόπο μπορούμε να χρησιμοποιήσουμε υπηρεσίες αναγνώρισης φωνής μεγάλου λεξιλογίου από συσκευές στις οποίες δεν υπάρχει η δυνατότητα απέυθείας αναγνώρισης. Τα δύο αυτά *contributions* δημιουργούν τις δυνατότητες για την ανάπτυξη εμπορικών εφαρμογών για κατανεμημένη αναγνώριση φωνής αν και κάτι τέτοιο απαιτεί αρκετές επεκτάσεις.

Πιθανές μελλοντικές επεκτάσεις λοιπόν μπορεί να είναι οι εξής :

- Χρήση ενός πιο πολύπλοκου τρόπου κωδικοποίησης όπως για παράδειγμα διανυσματικού κβαντισμού με μνήμη που να εκμεταλλεύεται την συσχέτιση μεταξύ των διαδοχικών *frames*.

- Επέκταση του *front-end* έτσι ώστε να υποστηρίζει *silence detection*. Με αυτό τον τρόπο θα κωδικοποιούνται και θα μεταδίδονται λιγότερα *frames*.

- Μελέτη της επίδρασης του θορύβου και της καθυστέρησης της μετάδοσης των πακέτων στην περίπτωση χρήσης ασύρματων δικτύων δεδομένων όπως *GPRS*.

- Χρήση σε πιο απλές συσκευές όπως κινητά με υποστήριξη προγραμματισμού σε *J2ME-CLDC-MIDP*. Αυτό θα μπορεί να γίνει από τη στιγμή που θα γίνει διαθέσιμο το *Mobile Multimedia API - MMAPI*, έτσι ώστε να είναι δυνατή η ηχογράφιση της φωνής κάτι που αυτή την στιγμή δεν είναι εφικτό.

Παράρτημα Α1

Α-1.1 Πειράματα bit allocation

Όπως είδαμε στα Κεφάλαια 2 και 4, ο διανυσματικός κβαντισμός με χρήση υποδιανυσμάτων (*product VQ*), μπορεί να δώσει πολύ καλά αποτελέσματα, αν επιλεχθούν κατάλληλα τα υποδιανύσματα (επιλογή συντελεστών και ποσότητα πληροφορίας που απαιτούν). Δείξαμε στο Κεφάλαιο 4 ότι η καλύτερη επίδοση που επιτύχαμε ήταν τα *2kbps*. Εδώ παρατίθενται μερικά από τα πειράματα που έγιναν κατά τη διάρκεια του *bit-allocation* (βλέπε παράγραφο 4.4.3), με 3-6 υποδιανύσματα (τα πειράματα 3, 3+, 4, 5, 5+ και 6). Τα πειράματα αυτά έδωσαν πολύ καλά αποτελέσματα, τα καλύτερα από τα οποία είδαμε στο Κεφάλαιο 4 και τα οποία δημοσιεύτηκαν στα [10], [11] και [12]. Όπως επισημάνθηκε στο Κεφάλαιο 4, ο *bit-allocation* αλγόριθμος μας βοηθάει στην εύρεση ενός αποτελεσματικού τρόπου διάθεσης των *bits* στα υποδιανύσματα, προσπαθώντας παράλληλα να διατηρήσει σε υψηλά επίπεδα τα αποτελέσματα της αναγνώρισης.

Γιά κάθε πείραμα υπάρχουν 2 πίνακες. Ο πίνακας στα αριστερά δείχνει τη δομή των υποδιανυσμάτων, ενώ στα δεξιά περιγράφεται η ακολουθία του *bit-allocation*. Όλα τα πειράματα ξεκινούν από 12 *bits* - πείραμα 1. Οι πρώτες επαναλήψεις αυξάνουν τον αριθμό *bits* ανά 2, ενώ οι επόμενες ανά 1. Έτσι φτάνουμε γρήγορα σε κάποια καλά αποτελέσματα, ενώ από εκεί και πέρα η διαδικασία γίνεται πιο ακριβής. Το καλύτερο αποτέλεσμα για κάθε επανάληψη (αυτό που φαίνεται έντονα) κατοχυρώνεται, παίρνοντάς το σαν βάση για την επόμενη επανάληψη. Πολλές φορές φτάνουμε γρήγορα στο μέγιστο αριθμό *bits* που έχουμε διαθέσει για την εκπαίδευση ενός υποδιανύσματος, οπότε για τις επόμενες επαναλήψεις, η διαδικασία συνεχίζεται για τα υπόλοιπα μόνο υποδιανύσματα.

Δεδομένης της γνώσης που αποκομίσαμε από τα αποτελέσματα των πειραμάτων (οι πρώτοι συντελεστές απαιτούν περισσότερη πληροφορία) θα προσπαθήσουμε να ερμηνεύσουμε τα πειράματα που ακολουθούν.

A-1.2 πείραμα 3

Υποδιανύσματα και πείραμα 1 :
Εδώ βλέπουμε ότι οι 3 πιο σημαντικοί συντελεστές (1-3 αντίστοιχα), είναι κατανεμημένοι στα 3 υποδιανύσματα. Λόγω του μεγέθους τους, το δεύτερο υποδιάνυσμα απαιτεί ένα *bit* παραπάνω από το πρώτο, ενώ το τρίτο απαιτεί 5 *bits* καθώς περιλαμβάνει επιπρόσθετα, τον αμέσως επόμενο από βαθμό σημαντικότητας συντελεστή (4).

Επαναλήψεις :
1. (πείραμα 2.1) : συντελεστής 1
2. (πείραμα 3.3) : συντελεστές 2, 4
3. (πείραμα 4.1) : συντελεστής 1 (μέγιστος αριθμός *bits*)
4. (πείραμα 5.3) : συντελεστές 2, 4
5. (πείραμα 6.3) : συντελεστές 2, 4
6. (πείραμα 7.3) : συντελεστές 2, 4

Παρατηρούμε ότι για το δεύτερο υποδιάνυσμα, ποτέ δεν χρειάστηκαν πάνω από 4 *bits*, πληροφορία ίση περίπου με αυτή που χρειάζεται μόνος του ο συντελεστής 3. Άλλωστε οι υπόλοιποι 4 συντελεστές του διανύσματος είναι οι λιγότερο σημαντικοί του *MFCC* διανύσματος. Έτσι, μόνο το 1ο και 3ο υποδιάνυσμα κερδίζουν *bits*. Δεδομένου ότι δεν είχαμε προβλέψει πάνω από 7 *bits* για το 1ο υποδιάνυσμα, όλα τα υπόλοιπα τα κερδίζει το 3ο υποδιάνυσμα.

SubVector	*elements*
1	1 5 10
2	3 8 9 12 13
3	2 4 6 7 11

exper	*% error*	*bits − allocated*	*total*
1		3 4 5	12
2.1	**14.96**	**5 4 5**	14
2.2	16.94	3 6 5	
2.3	15.42	3 4 7	
3.1	12.14	7 4 5	16
3.2	13.36	5 6 5	
3.3	**11.58**	**5 4 7**	
4.1	**9.12**	**7 4 7**	18
4.2	10.77	5 6 7	
4.3	10.67	5 4 9	
5.2	8.59	7 5 7	19
5.3	**8.48**	**7 4 8**	
6.2	8.69	7 5 8	20
6.3	**8.31**	**7 4 9**	
7.2	8.23	7 5 9	21
7.3	**7.82**	**7 4 10**	

A-1.3 πείραμα 3+

Υποδιανύσματα και πείραμα 1 :
Εδώ βλέπουμε ότι οι 3 πιο σημαντικοί συντελεστές (1-3 αντίστοιχα), είναι όλοι στο πρώτο υποδιάνυσμα, οι αμέσως 5 πιο σημαντικοί στο 2ο και οι υπόλοιποι επίσης 5 στο 3ο. Όλα τα υποδιανύσματα είναι αρχικά με 4 *bits*. Ήδη με τα όσα έχουμε δει έως τώρα προβλέπουμε ότι ο τελικός αριθμός *bits* θα είναι μεγαλύτερος από το 1ο στο 3ο υποδιάνυσμα με κίνδυνο μάλιστα να μην φτάσουν πάλι τα *bits* για το 1ο υποδιάνυσμα.

Επαναλήψεις :
1. (πείραμα 2.1) : συντελεστές 1, 2, 3
2. (πείραμα 3.2) : λόγω μεγέθους
3. (πείραμα 4.2) : λόγω μεγέθους
4. (πείραμα 5.1) : συντελεστές 1, 2, 3
5. (πείραμα 6.1) : συντελεστές 1, 2, 3 (μέγιστος αριθμός *bits*)
6. (πείραμα 7.3) : υποδιάνυσμα 3 (ένα *bit* μόνο)
7. (πείραμα 8.3) : υποδιάνυσμα 3 (εντάξει τώρα)

Παρατηρούμε ότι οι προβλέψεις βγήκαν αληθινές!

SubVector	*elements*
1	1 2 3
2	4 5 6 7 8
3	9 10 11 12 13

exper	*% error*	*bits − allocated*	*total*
1		4 4 4	12
2.1	**11.43**	**6 4 4**	14
2.2	11.63	4 6 4	
2.3	14.19	4 4 6	
3.1	9.53	8 4 4	16
3.2	**9.19**	**6 6 4**	
3.3	10.39	6 4 6	
4.1	8.46	8 6 4	18
4.2	**8.38**	**6 8 4**	
4.3	8.56	6 6 6	
5.1	**7.62**	**8 7 4**	19
5.2	7.90	6 9 4	
5.3	8.48	6 7 6	
6.1	**7.54**	**9 7 4**	20
6.2	7.62	8 8 4	
6.3	7.57	6 7 5	
7.2	7.65	9 8 4	21
7.3	**7.44**	**9 7 5**	
8.2	7.42	9 8 5	22
8.3	**7.32**	**9 7 6**	

A-1.4 πείραμα 4

Υποδιανύσματα και πείραμα 1 :
Εδώ βλέπουμε ότι οι 4 πιο σημαντικοί συντελεστές (1-4 αντίστοιχα), είναι κατανεμημένοι στα 4 υποδιανύσματα, ενώ το 1ο και 2ο υποδιάνυσμα έχουν μέγεθος μόλις δυο, αφού περιέχουν τους αμέσως πιο σημαντικούς συντελεστές (5 και 6 αντίστοιχα). Το 1ο πείραμα κατανέμει 2, 4, 2 και 4 *bits* αντίστοιχα στα 4 υποδιανύσματα. Προβλέπουμε πάλι, ότι ο τελικός αριθμός *bits* για το 1ο και 4ο υποδιάνυσμα μάλλον θα ναι ο μεγαλύτερος, αφού αυτά περιέχουν τους 2 πιο σημαντικούς συντελεστές (1 και 2).

Επαναλήψεις :
1. (πείραμα 2.1) : συντελεστής 1
2. (πείραμα 3.3) : συντελεστές 4, 6
3. (πείραμα 4.4) : συντελεστής 2
4. (πείραμα 5.1) : συντελεστής 1 (μέγιστος αριθμός *bits*)
5. (πείραμα 6.4) : συντελεστής 2
6. (πείραμα 7.3) : συντελεστές 4, 6

Παρατηρούμε ότι για το 1ο υποδιάνυσμα έχουμε για άλλη μια φορά έλλειψη *bits*!

SubVector	*elements*
1	1 5
2	3 8 9 12 13
3	4 6
4	2 7 10 11

exper	% *error*	$bits - allocated$	*total*
1		2 4 2 4	12
2.1	**15.70**	**4 4 2 4**	14
2.2	20.19	2 6 2 4	
2.3	17.63	2 4 4 4	
2.4	19.10	2 4 2 6	
3.1	12.45	6 4 2 4	16
3.2	14.01	4 6 2 4	
3.3	**11.26**	**4 6 4 4**	
3.4	12.37	4 6 2 6	
4.1	9.80	6 4 4 4	18
4.2	10.75	4 6 4 4	
4.3	10.87	4 4 6 4	
4.4	**9.55**	**4 4 4 6**	
5.1	**8.81**	**6 4 4 5**	19
5.2	9.55	4 6 4 5	
5.3	9.55	4 4 6 5	
5.4	8.86	4 4 4 7	
6.2	8.66	6 5 4 5	20
6.3	8.69	6 4 5 5	
6.4	**8.36**	**6 4 4 6**	
7.2	8.15	6 5 4 6	21
7.3	**8.03**	**6 4 5 6**	
7.4	8.10	6 4 4 7	

A-1.5 πείραμα 5

Υποδιανύσματα και πείραμα 1 :
Εδώ απλά έχουμε ένα παραπάνω υποδιάνυσμα, με τους συντελεστές 8 και 10 από τα 2ο και 4ο αντίστοιχα υποδιάνυσμα του προηγούμενο πειράματος και περιμένουμε αντίστοιχη συμπεριφορά - αποτελέσματα.
Επαναλήψεις :
1. (πείραμα 2.1) : συντελεστής 1
2. (πείραμα 3.3) : συντελεστές 4, 6
3. (πείραμα 4.4) : συντελεστής 2
4. (πείραμα 5.4) : συντελεστής 2
5. (πείραμα 6.4) : συντελεστές 2 (μέγιστος αριθμός *bits*)
6. (πείραμα 7.1) : συντελεστής 1 (ένα μόνο *bit*)
7. (πείραμα 8.1) : συντελεστής 1 (εντάξει τώρα)

SubVector	*elements*
1	1 5
2	3 9 12 13
3	4 6
4	2 7 11
5	8 10

exper	*% error*	*bits − allocated*	*total*
1		2 3 2 3 2	12
2.1	**18.77**	**4 3 2 3 2**	14
2.2	23.83	2 5 2 3 2	
2.3	20.75	2 3 4 3 2	
2.4	19.99	2 3 2 5 2	
2.5	25.91	2 3 2 3 4	
3.1	15.21	6 3 2 3 2	16
3.2	15.06	4 5 2 3 2	
3.3	**13.36**	**4 3 4 3 2**	
3.4	13.36	4 3 2 5 2	
3.5	16.18	4 3 2 3 4	
4.1	10.28	6 3 4 3 2	18
4.2	11.32	4 5 4 3 2	
4.3	12.9	4 3 6 3 2	
4.4	**10.24**	**4 3 4 5 2**	
4.5	12.22	4 3 4 3 4	
5.1	9.02	6 3 4 4 2	19
5.2	9.32	4 5 4 4 2	
5.3	10.15	4 3 6 4 2	
5.4	**8.92**	**4 3 4 6 2**	
5.5	9.88	4 3 4 4 4	
6.1	8.69	6 3 4 5 2	20
6.2	8.93	4 5 4 5 2	
6.3	9.93	4 3 6 5 2	
6.4	**8.38**	**4 3 4 7 2**	
6.5	9.52	4 3 4 5 4	
7.1	**7.72**	**5 3 4 7 2**	21
7.2	8.07	4 4 4 7 2	
7.3	8.45	4 3 5 7 2	
7.5	8.20	4 3 4 7 3	
8.1	**7.01**	**6 3 4 7 2**	22
8.2	7.87	5 4 4 7 2	
8.3	7.57	5 3 5 7 2	
8.5	7.70	5 3 4 7 3	

A-1.6 πείραμα 5+

Υποδιανύσματα και πείραμα 1 :
Εδώ βλέπουμε μια κατά σειρά σημαντικότητας κατανομή των συντελεστών(όπως και στο πείραμα 3+) σε 5 υποδιανύσματα, με τα 3 τελευταία να έχουν μεγαλύτερο μέγεθος, αφού περιέχουν λιγότερο σημαντικούς συντελεστές. Φαίνεται, ότι τα πολύ καλά αποτελέσματα οφείλονται και στην καλύτερη συσχέτιση (αν και μικρή), που υπάρχει ανάμεσα στους συντελεστές των υποδιανυσμάτων που επιλέχθηκαν.

Επαναλήψεις :
1. (πείραμα 2.1) : συντελεστές 1 και 2
2. (πείραμα 3.3) : συντελεστές 5, 6, 7 (μέγεθος 3)
3. (πείραμα 4.2) : συντελεστές 3 και 4 !
4. (πείραμα 5.4) : συντελεστές 8, 9, 10 (μέγεθος 3)
5. (πείραμα 6.2) : συντελεστές 3 και 4 (είχαν πάρει μόνο 1 *bit*)

Το γεγονός ότι αυτή τη φορά δεν έχουμε έλλειψη *bits*, καθώς και η καλή αρχική κατανομή κατανομή τους, μας δίνει σε μόλις 5 επαναλήψεις (*2Kbps*) αποτέλεσμα *6.63*, πολύ κοντά στο βέλτιστο *6.55*!

SubVector	*elements*
1	1 2
2	3 4
3	5 6 7
4	8 9 10
5	11 12 13

exper	*% error*	*bits − allocated*	*total*
1	16.79	3 3 2 2 2	12
2.1	**11.71**	**5 3 2 2 2**	14
2.2	13.13	3 5 2 2 2	
2.3	11.96	3 3 4 2 2	
2.4	13.72	3 3 2 4 2	
2.5	13.84	3 3 2 2 4	
3.1	11.71	7 3 2 2 2	16
3.2	9.50	5 5 2 2 2	
3.3	**9.30**	**5 3 4 2 2**	
3.4	9.40	5 3 2 4 2	
3.5	10.08	5 3 2 2 4	
4.1	8.38	7 3 4 2 2	18
4.2	**7.80**	**5 5 4 2 2**	
4.3	8.43	5 3 6 2 2	
4.4	8.10	5 3 4 4 2	
4.5	8.69	5 3 4 2 4	
5.1	8.38	7 4 4 2 2	19
5.2	7.80	5 6 4 2 2	
5.3	7.81	5 4 6 2 2	
5.4	**6.99**	**5 4 4 4 2**	
5.5	7.95	5 4 4 2 4	
6.1	7.06	6 4 4 4 2	20
6.2	**6.63**	**5 5 4 4 2**	
6.3	6.96	5 4 5 4 2	
6.4	6.83	5 4 4 5 2	
6.5	7.16	5 4 4 4 3	

Α-1.7 πείραμα 6

Υποδιανύσματα και πείραμα 1 :
Εδώ έχουμε 5 υποδιανύσματα με 2 συντελεστές και ένα με 3. Λόγω και του μεγάλου αριθμού υποδιανυσμάτων είναι δυσκολότερο να φτάσουμε με λίγες μόνο επαναλήψεις σε καλά αποτελέσματα, ειδικά αν δει κανείς ότι η αρχική κατανομή των *bits* δεν είναι και η καλύτερη. Έτσι, στα 20 *bits* το σφάλμα αναγνώρισης είναι ακόμα στο 8.10!

Επαναλήψεις :
1. (πείραμα 2.4) : λόγω συντελεστή 2
2. (πείραμα 3.1) : λόγω συντελεστή 1
3. (πείραμα 4.3) : συντελεστές 4 και 6
4. (πείραμα 5.3) : συντελεστές 4 και 6 (για ελάχιστη διαφορά με 5.2 πείραμα)
5. (πείραμα 6.2) : λόγω συντελεστή 3 (επιτέλους ...)
6. (πείραμα 7.1) : λόγω συντελεστή 1 (μέγιστος αριθμός *bits*)
7. (πείραμα 8.4) : λόγω συντελεστή 2
8. (πείραμα 9.2) : λόγω συντελεστή 3

Λόγω μεγέθους του πίνακα τα αποτελέσματα φαίνονται στην επόμενη σελίδα.

Α-1.8 Σύγκριση αποτελεσμάτων bit allocation

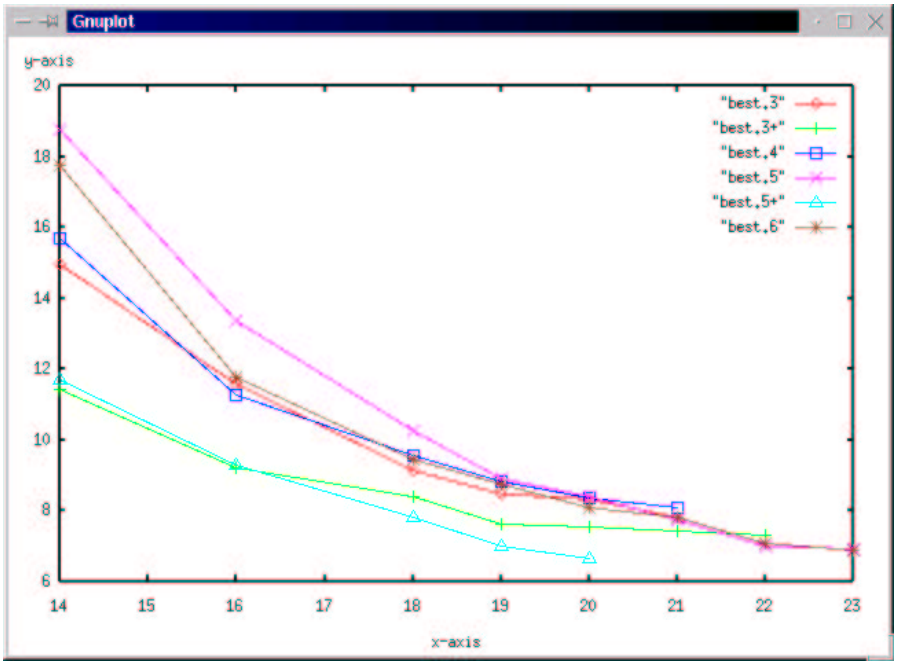
Είδαμε στο Κεφάλαιο 4, ότι η σημαντικότητα των συντελεστών παίζει πρωταρχικό ρόλο. Παρ'όλα αυτά, κανείς θα μπορούσε να χρησιμοποιήσει αυτό σαν πρωτεύον κριτήριο, με δευτερεύον κριτήριο τις συσχετίσεις μεταξύ των συντελεστών, αποκομίζοντας έτσι ακόμα μεγαλύτερα οφέλη. Φυσικά, μεγάλο ρόλο στη διεξαγωγή των πειραμάτων, διαδραματίζει και η αρχική κατανομή των *bits* στα υποδιανύσματα. Αν αυτή δεν είναι η κατάλληλη, όσο καλές και αν είναι οι υπόλοιπες επιλογές, είναι δύσκολο να καταλήξουμε σε καλά αποτελέσματα, σε λίγες μόνο επαναλήψεις. Επιπρόσθετα, το γεγονός ότι οι πρώτες επαναλήψεις ήταν με βήμα 2 *bits*, ίσως συνετέλεσε αρνητικά σε κάποιες περιπτώσεις. Τέλος, το γεγονός ότι κάποια *codebooks* αποδείχθηκαν μικρότερα από ότι είχε προβλεφτεί, συνετέλεσε στην όχι τόσο καλή απόδοση ή και στην αργή σύγκλιση σε κάποιο καλό αποτέλεσμα.

Η σύγκριση των αποτελεσμάτων φαίνεται στο Σχήμα Α-1. Φαίνεται καθαρά η υπεροχή του πειράματος 5+, το οποίο φτάνει κοντά στο βέλτιστο 6.55, σε μόλις 4 επαναλήψεις. Τα πειράματα 6 και 5 καταφέρνουν να φτάσουν σε απόδοση 6.78 και 6.93 αντίστοιχα, σε 8 όμως επαναλήψεις - 23 *bits*. Ακολου-

θεί το πείραμα 3+, με απόδοση 7.32 στα 22 *bits*, ενώ τα πειράματα 3 και 4 δείχνουν σχετικά απογοητευτικά, με απόδοση μόλις 7.82 και 8.10 αντίστοιχα στα 21 *bits*.

SubVector	*elements*
1	1 5
2	3 9
3	4 6
4	2 7
5	8 10
6	11 12 13

exper	*% error*	*bits − allocated*	*total*
1		2 2 2 2 2 2	12
2.1	20.96	4 2 2 2 2 2	14
2.2	25.68	2 4 2 2 2 2	
2.3	22.83	2 2 4 2 2 2	
2.4	**17.75**	**2 2 2 4 2 2**	
2.5	29.46	2 2 2 2 4 2	
2.6	29.41	2 2 2 2 2 4	
3.1	**11.76**	**4 2 2 4 2 2**	16
3.2	14.10	2 4 2 4 2 2	
3.3	12.80	2 2 4 4 2 2	
3.4	15.70	2 2 2 6 2 2	
3.5	16.00	2 2 2 4 4 2	
3.6	16.43	2 2 2 4 2 4	
4.1	10.34	6 2 2 4 2 2	18
4.2	10.01	4 4 2 4 2 2	
4.3	**9.45**	**4 2 4 4 2 2**	
4.4	10.42	4 2 2 6 2 2	
4.5	11.66	4 2 2 4 4 2	
4.6	11.28	4 2 2 4 2 4	
5.1	9.07	6 2 3 4 2 2	19
5.2	8.79	4 4 3 4 2 2	
5.3	**8.76**	**4 2 5 4 2 2**	
5.4	9.09	4 2 3 6 2 2	
5.5	9.60	4 2 3 4 4 2	
5.6	9.60	4 2 3 4 2 4	
6.1	8.36	6 2 4 4 2 2	20
6.2	**8.10**	**4 4 4 4 2 2**	
6.3	9.17	4 2 6 4 2 2	
6.4	8.56	4 2 4 6 2 2	
6.5	9.12	4 2 4 4 4 2	
6.6	9.25	4 2 4 4 2 4	
7.1	**7.82**	**6 3 4 4 2 2**	21
7.2	8.10	4 5 4 4 2 2	
7.3	8.26	4 3 6 4 2 4	
7.4	8.10	4 3 4 6 2 2	
7.5	8.38	4 3 4 4 4 2	
7.6	8.26	4 3 4 4 2 4	
8.2	7.19	6 4 4 4 2 2	22
8.3	7.34	6 3 5 4 2 2	
8.4	**7.09**	**6 3 4 5 2 2**	
8.5	7.26	6 3 4 4 3 2	
8.6	8.18	6 3 4 4 2 3	
9.2	**6.78**	**6 4 4 5 2 2**	23
9.3	7.14	6 3 5 5 2 2	
9.4	7.26	6 3 4 6 2 2	
9.5	6.86	6 3 4 5 3 2	
9.6	6.96	6 3 4 5 2 3	



Σχήμα Α-1: **Σύγκριση αποτελεσμάτων** Στο σχήμα φαίνονται και τα 6 πειράματα. Ο y-άξονας είναι το $WER(\%)$ και ο x-άξονας ο συνολικός αριθμός *bits*. Το πείραμα 5+ είναι μακράν η καλύτερη επιλογή δίνοντας WER 6.63 στα μόλις 20 *bits* ($2Kbps$)!

Καταλήγοντας, βλέπουμε ότι για την καλή απόδοση των πειραμάτων, ένας μεγάλος αριθμός παραγόντων πρέπει να καθοριστεί κατάλληλα, κάτι αρκετά δύσκολο. Παρ'όλα αυτά, καταφέραμε να πάρουμε πολύ καλά αποτελέσματα με μόλις 2 $Kbps$, κάτι εξαιρετικά ενθαρρυντικό. Είναι πράγματι μια πρόκληση, να δούμε αν και κατά πόσο ακόμα, μπορεί να μειωθεί αυτός ο ήδη πολύ χαμηλός ρυθμός μετάδοσης.

Βιβλιογραφία

[1] Steve Young : *Large Vocabulary Continuous Speech Recognition: A Review,* April 1996.

[2] Tony Robinson : *Speech Analysis*, lecture notes.

[3] *The HTK Book*, Entropic, Cambridge Research Laboratory.

[4] Lawrence Rabiner, Biing-Hwang Juang : *Fundamentals of Speech Recognition,* Prentice Hall.

[5] John R. Deller, John G. Proakis, John H. L. Hansen : *Discrete Processing of Speech Signals* , Prentice Hall.

[6] John Makhoul, Salim Roucos, Herbert Gish : *Vector Quantization in Speech Coding,* Proceedings of the IEEE, VOL 73, NO 11, November 1985.

[7] Robert M. Gray : *Vector Quantiaztion,* IEEE ASSR Magazine April 1984.

[8] Allen Gersho, Robert M. Gray : *Vector Quantization And Signal Compression,* Kluwer Academic Publishers.

[9] John R. Deller, John G. Proakis, Hohn H. L. Hansen : *Discrete-Time Processing of Speech Signals*, Prentice Hall

[10] V.Digalakis, L.Neumeyer, M.Perakakis : *Product-code Vector Quantization of Cepstral Parameters for Speech Recognition over the Web*, Proceedings ICSLP, December 1998.

[11] V.Digalakis, L.Neumeyer, M.Perakakis : *Quantization of Cepstral Parameters for Speech Recognition over the Web* Proceedings ICCASP '98, Seattle, WA, May 1998.

[12] V.Digalakis, L.Neumeyer, M.Perakakis : *Quantization of Cepstral Parameters for Speech Recognition over the Web* IEEE Journal on Selected Areas In Communications, Jan. 1999, volume 17, pp. 82-90.

[13] Hakon Gudding, *Capacity Analysis of GPRS*, Master Thesis

[14] David Pearce - Motorola Labs & Chairman ETSI STQ-Aurora DSR Working Group, *Enabling New Speech Driven Services for Mobile Devices : An overview of the ETSI standards activitiew for Distributed Speech Recognition Front-ends*

[15] *Java 2 Platform Micro Edition(J2ME) for Creating Mobile Devices*, White Paper, Sun microsystems

[16] *Mobile Applications Initiative : Developer's Guide, www.mai.com*, Ericsson

[17] *Wirelles Packet Data and its Applications*, Symbian Conference Transcript, Symbian Developer Network(www.symbiandevnet.com)

[18] Τσακαλίδης Σταύρος, Διπλωματική εργασία : *Κρυφά μαρκοβιανά μοντέλα με μείγματα διακριτών κατανομών για αναγνώριση ομιλίας μεγάλου λεξιλογίου.*