NATURAL SOUNDING TEXT-TO-SPEECH SYNTHESIS BASED ON SYLLABLE-LIKE UNITS

A THESIS

submitted by

SAMUEL THOMAS

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

MAY 2007

THESIS CERTIFICATE

This is to certify that the thesis entitled **Natural Sounding Text-to-Speech Synthesis based on Syllable-like Units** submitted by **Samuel Thomas** to the Indian Institute of Technology Madras for the award of the degree of Master of Science (by Research) is a bonafide record of research work carried out by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Hema A. Murthy

Chennai-600 036 Dr. C. Chandra Sekhar

Date: Dept. of Computer Science and Engg.

ACKNOWLEDGMENTS

There are a large number of people to thank for helping me do the work in this thesis. Without them, I would have been no where and would have as well given up long back! I am very grateful to my guide Prof. Hema Murthy, for allowing me to work in the area of speech synthesis. Madam has always been there to listen to my problems, related to research and otherwise, and to help me with the best she could. I am extremely grateful for her guidance and all the help she has extended over the days. My co-guide Prof. Chandra Sekhar has been very helpful to me in doing this work. I am very grateful to him for helping me improve my writing skills with the suggestions he gave after reading my reports. I am extremely grateful for his guidance and help. I would also like to thank my project coordinator Prof. C.S. Ramalingam for his guidance, advice and deadlines while working on the projects related to this work. I am also grateful to the Prof. Timothy Gonsalves, Head of Department, Computer Science and Engineering and the department staff for providing all the facilities and help to do this work.

Many other people deserve my whole hearted thanks for making this work possible. I am extremely grateful to my colleague Nageshwara Rao for helping me all through the work in building the speech synthesizers. I would also like to thank my lab mates at the MMI Lab, Venu, Swetha, Jithendra, Kasirajan and Sree Hari for the special moments we had at the lab. Thank you very much! Thanks must also go to all my colleagues in the speech group especially Deivapalan and Sreenivas for their help with the work on embedded devices. My special thanks also goes to the speech teams at Lattice

Bridge Technologies, Chennai and Amdale Technologies, New Delhi for helping with the speech data and annotation work. Everybody in the DON Lab needs a mention for all help. Thank you!

I am ever grateful to my special friends, Dinil, Nisha and Vivek for all the help and love they have shown. I have learnt a lot from them and cherish each moment we had together. Thank you very much! I am grateful also to Lakshmi, Tarun and Hima for always being there to help. Thank you very much!

I would never have started or completed this work if I didn't have a loving family to encourge me and keep me going. Words cannot express my thanks. Thank you very much!

ABSTRACT

Keywords: *Text-to-speech synthesis, Concatenative speech synthesis, Unit selection based speech synthesis, Group delay based speech segmentation, Distributed speech synthesis.*

Many text-to-speech synthesizers for Indian languages have used synthesis techniques that require prosodic models for good quality synthetic speech. However, due to unavailability of adequately large and properly annotated databases for Indian languages, prosodic models for these synthesizers have still not been developed properly. With inadequate prosodic models in place, the quality of synthetic speech generated by these synthesizers is poor. In this work, we develop Indian language speech synthesizers that do not require extensive prosodic models by using a new "syllable-like" speech unit suitable for concatenative speech synthesis.

The syllable-like units are automatically generated using a group delay based segmentation algorithm and acoustically correspond to the form C*VC* where C is a consonant, V is a vowel and C^* indicates the presence of 0 or more consonants. The effectiveness of the unit is demonstrated by using these units to build a unit selection synthesizer, based on the Festival speech synthesis framework, for Tamil and Hindi. The unit selection technique currently implemented in Festival uses an automatic clustering technique to first cluster units based on their phonetic and prosodic context. Units that minimize acoustically defined target and join costs are then selected from a cluster. Evaluations are done to ascertain the improvement in the quality of synthesized speech using the syllable-like unit, the order in which syllable-like units need to be concatenated together, how much of useful duration information is intrinsically present within the syllable-like unit and how the speech units can be used for unrestricted speech synthesis. Since the storage/memory requirement of concatenative speech synthesizers is fairly high, an analysis is also done on how to reduce this requirement. Significant improvement in quality is obtained when bisyllable units are used, instead of using monosyllables only, with results better than in the conventional diphone-based approach. The naturalness of the synthesized speech demonstrates the appropriateness of the proposed approach.

This work has also addressed the issues in building speech synthesizers for embedded devices. Distributed speech synthesis(DSS) has been considered as a means by which unrestricted speech synthesis can be made available on such devices. The architectural design and implementation for distributed speech synthesis systems based on two synthesizers - Festival and Flite, have been presented in the work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Text-to-speech synthesis

One of the most distinct features of human civilizations that have evolved along with them over time is probably the language. Speech holds an important role in this evolution by not only being the spoken form of a language but also the most efficient way of communication. Probably because of its profound influence on day to day human lives, speech has been a subject of constant research for centuries. Various aspects of speech - recognition, synthesis, understanding and identification have been studied to see how these tasks, that humans do effortlessly, can be done by machines.

The goal of text-to-speech synthesis (TTS) is the automatic conversion of unrestricted natural language sentences in text form to a spoken form that closely resembles the spoken form of the same text by a native speaker of the language. This field of speech research has witnessed significant advances over the past decade with many systems being able to generate a close to natural sounding synthetic speech. Research in the area of speech synthesis has been fueled by the growing importance of many new applications. These include information retrieval services over telephone such as banking services, public announcements at places like train stations and reading out manuscripts for collation. Speech synthesis has also found applications in tools for reading emails, faxes and web pages over telephone and voice output in automatic translation systems. Special equipment for the physically challenged, such as word processors with reading-out capability and book-reading aids for visually challenged and speaking aids for the vocally challenged also use speech synthesis [1].

1.2 Issues and approaches in text-to-speech synthesis

The important challenge that needs to be addressed in any TTS systems is how to produce natural sounding speech for a plain text given as its input. It is not feasible to solve this by recording and storing all the words of a language and then concatenating the words in the given text to produce the corresponding speech. The TTS systems first convert the input text into its corresponding linguistic or phonetic representations and then produce the sounds corresponding to those representations. With the input being a plain text, the generated phonetic representations also need to be augmented with information about the intonation and rhythm that the synthesized speech should have. This task is done by a text analysis module in most speech synthesizers. The transcription from the text analysis module is then given to a digital signal processing (DSP) module that produces synthetic speech [2]. Figure 1.1 shows the block diagram of a TTS system.

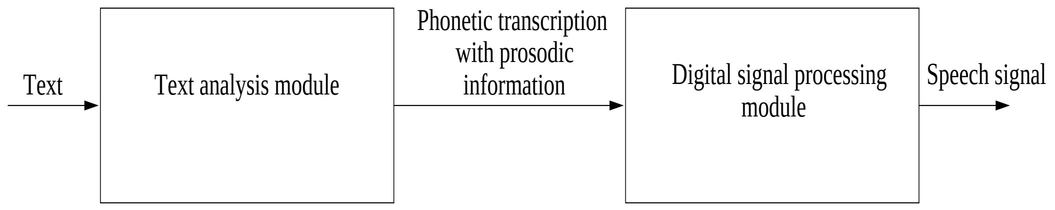
Text
Text analysis module
Phonetic transcription with prosodic information
Digital signal processing module
Speech signal

Fig. 1.1: Block diagram of a TTS system

1.2.1 Text analysis module

Conversion of plain text into its linguistic representation along with prosodic information is highly language dependent. In some languages, like Indian languages, the transformation is simple because the scripts are orthographic representations of speech sounds. However, in languages like English, the conversion is not straight forward and these languages need large sets of pronunciation rules. Homographs that have the same spellings with different meanings and pronunciations depending on the context are examples of words that need detailed processing. The text analysis task can be divided into three subtasks [2] as follows:

- Syntactic processing has to be done on the input text to obtain a phonetic transcription. In this stage, the sentences in the text are divided into words and the numbers, abbreviations and acronyms are expanded. This is commonly done by using regular grammars. Possible part-of-speech categories for each word are then searched to find the probable classes based on the contextual information of words using techniques based on finite state automata, neural networks or classification and regression trees (CART) [3].

- Once ambiguities in the text have been sorted out, automatic determination of phonetic or linguistic transcription of the text is performed using either dictionary-based or rule-based strategies.

- Predicting the correct intonation (how the pitch pattern or fundamental frequency changes during speech), stress and duration from the plain text is a challenging task. The prosodic or suprasegmental features also depend on many aspects like the speaker characteristics (gender, age), emotions and meaning of the sentence (neutral, imperative or question) [4]. To perform this task, syntactic-prosodic structures of sentences corresponding to clause and phrase constituents are first generated. These structures are then used to predict the durations of speech units and the intonation to be applied to the units. This step requires phonetic and phonological knowledge obtained from experts or automatically acquired using statistical methods like CART.

1.2.2 Digital signal processing module

Given a phonetic transcription along with prosodic information, the DSP module tries to mimic the human speech production system to produce speech. Commonly used methods are:

- Articulatory synthesis that attempts to model the human speech production system directly [5],
- Formant synthesis that models the resonances of the vocal tract [6],
- Concatenative synthesis that uses prerecorded samples derived from natural speech [4].

The first two techniques are also referred to as rule-based synthesis techniques. Articulatory synthesis typically involves models of the articulators and vocal cords. The articulators are usually modeled with a set of area functions of the vocal tract between glottis and mouth. The main constraint with this method is that the parts of the human vocal tract like tongue are so complicated that it is almost impossible to model them precisely. Formant synthesis is based on the source-filter model of speech and describes speech in terms of upto 60 parameters, such as formant frequencies, anti-formant frequencies and their bandwidths. These parameters are determined using rules to synthesize a desired utterance.

Naturalness of synthetic speech produced by state-of-the art speech synthesis systems is mainly attributed to the use of concatenative speech synthesis [7] that uses phonemes, diphones, syllables, words or sentences as basic speech units. Text is synthesized by selecting appropriate units from a speech database and concatenating them. The quality of synthesized speech is influenced by the continuity of acoustic features (spectral envelope, amplitude, fundamental frequency, speaking rate) across concatenation points and the availability of appropriate units with proper prosodic features in the database [4]. If large units such as phrases or sentences are stored and used, the quality (intelligibility and naturalness) of synthesized speech is good, although the domain of synthesis is not unrestricted text. On the other hand, when small units such as phonemes are used, a wide range of words or sentences can be synthesized but with poor speech quality. In order to improve the quality of synthetic speech, synthesis methods using subword units as the basic sound units are employed [4]. The most commonly used subword unit is a diphone. Diphones provide a balance between context dependency and size (typically 1000-2000 in a language). Even though diphone synthesizers produce a reasonable quality speech, the prosody (pitch and duration) of each phone in the concatenated waveform does not correspond to the desired prosody. Several signal processing techniques have been developed for improving the prosody. Typical examples are the Time Domain Pitch Synchronous Overlap Add (TD-PSOLA) method [8] and the Harmonic plus Noise Model (HNM) method [9]. However, while diphone concatenation can produce a reasonable quality speech, a single example of each diphone is not enough to produce good quality speech.

A second approach for concatenative speech synthesis, namely, unit selection based concatenative synthesis, attempts to address these issues by collecting several examples of each unit with different values of pitch, duration and linguistic content so that the chosen unit is close to the target. The unit selection is based on two cost functions. The target cost, $C^t(u_i, t_i)$, is an estimate of the difference between a unit u_i in the database and the target, t_i. The concatenation cost, $C^c(u_{i-1}, u_i)$, is an estimate of the quality of concatenation between two consecutive units $/u_{i-1}/$ and $/u_i/$ [10]. The method chooses the examples of units that minimize both the costs. The target and concatenation costs are based on measures of phonetic features (identity of a speech unit, position of a speech unit, previous and next units, etc.) and prosodic features (F_0, Mel frequency cepstrum coefficients, power, etc.) [11]. The quality of the synthesized speech is a function of units available in the database. For good quality synthesis, examples of all units of the language should be present. Moreover, the units should also be generic so that they can be used for unrestricted synthesis.

Figure 1.2 shows the text-to-speech synthesis cycle in most TTS systems. In this cycle, the text preprocessing transforms the input text into a regularized format that can be processed by the rest of the system. This includes breaking the input text into sentences, tokenizing them into words and expanding the numerals. The prosodic phrasing component then divides the preprocessed text into meaningful chunks of information based on language models and constructs. The pronunciation generation component is responsible for generating the acoustic sequence needed to synthesize the input text by finding the pronunciation of individual words in the input text. Duration value for each segment of speech is determined by the segmental duration generation component. The function of the intonation generation component is to generate a fundamental frequency (F_0) contour for the input text to be synthesized. The waveform generation component takes as input the phonetic and prosodic information generated by the various components described above, and generates the speech output.

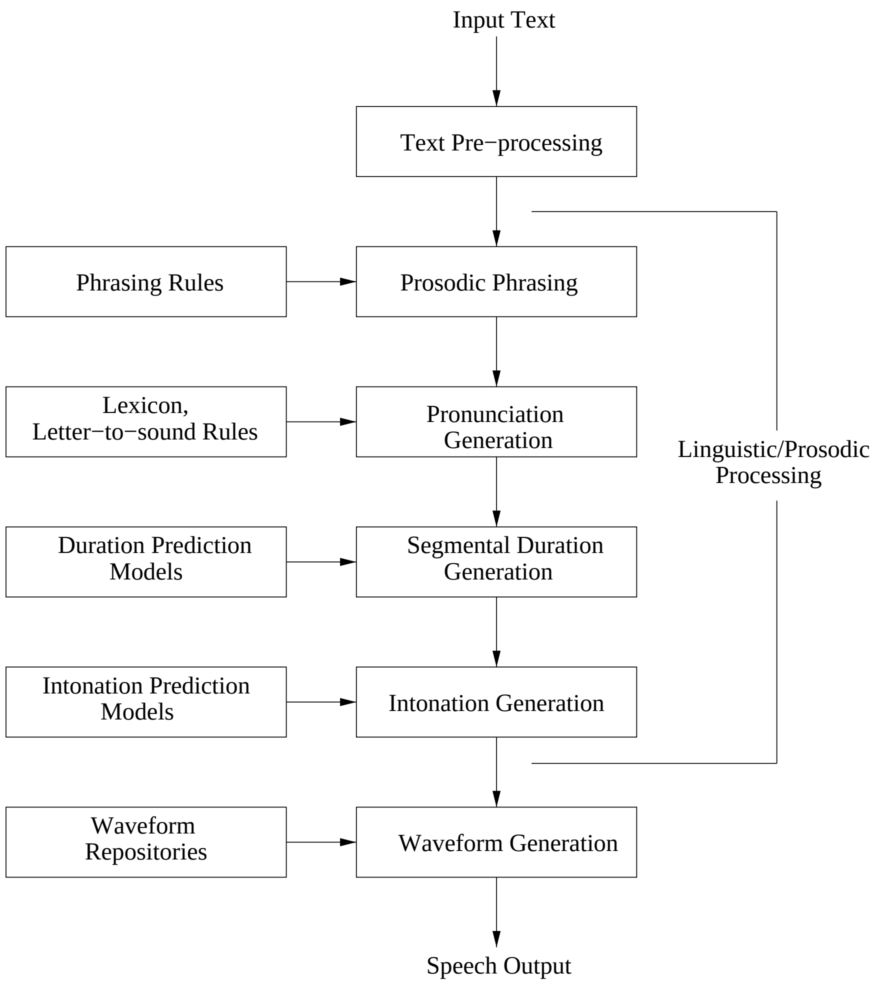

Fig. 1.2: Text-to-speech synthesis cycle

1.3 Text-to-speech synthesizers for Indian languages

Indian languages can be broadly categorized into two classes of languages - Dravidian and Aryan. The north Indian languages belong to the Aryan class of languages and the south Indian languages belong to the Dravidian class. A character in Indian languages is close to a syllable and can be typically of the form C^*VC^* where C is a consonant, V is a vowel and C^* indicates the presence of 0 or more consonants. There are about 35 consonants and 18 vowels in Indian languages [12].

Several works have been reported in developing TTS systems for Indian languages.

Rajeshkumar [13] and Yegnanarayana, *et al.*, [14] have used CV and CCV units as basic units in parameter based speech synthesis. Sen and Samudravijaya [15] have used formant synthesis techniques. Rama, *et al.*, [16] have reported a syllable based concatenative speech synthesizer similar to diphone based synthesis for Tamil with pitch modifications to obtain the desired pitch contours. Kishore, *et al.*, [17] have worked on a data driven synthesis method using syllables as the basic units and a unit selection algorithm to pick the best units. Sridhar and Murthy [18] have developed a diphone based speech synthesizer for Telugu and Hindi using the Festvox framework with prosody modeling.

1.4 Objective and scope of the thesis

Commonly used speech units for concatenative speech synthesis are phonemes, diphones, syllables, words or sentences. Most of the speech synthesizers available for Indian languages use subword units like diphones with a single instance of each unit for synthesis. Signal processing techniques are then used to modify the acoustics of the units to obtain the desired prosodic variations. The synthetic speech produced by these synthesizers is intelligible but not natural sounding. This is primarily because there are large number of concatenation points in the synthetic speech, leading to a formant trajectory that is not consistent. Diphone based synthesis also requires a significant amount of prosody modeling for duration, intonation and energy which in turn requires analysis of a voluminous amount of data and deduction of proper rules from the data. These efforts are both time consuming and laborious (described in detail in Chapter 2). With no such analysis in place, for many Indian languages it would be of great advantage if a speech unit that could produce synthetic speech with a few number of concatenation points and intrinsically has sufficient prosodic information in it, could be identified. This could reduce the need for prosody based analysis and result in faster development of TTS systems for Indian languages. Phonemes are not suitable for speech synthesis because they fail to model the dynamics of speech sounds with their large variability depending on the context [19]. Therefore, it is necessary to look for larger units like syllables for synthesis. Syllables are of longer duration and it has been observed that syllables are less dependent on the speaking rate variations than that of phonemes. The human auditory system integrates time spans of 200 milliseconds of speech that roughly corresponds to the duration of syllables [20]. Moreover, syllables also capture the co-articulation effects between sounds better than the phonemes [21]. For Indian languages, it is also seen that the syllable is a better choice than units like diphone or phone [22]. The primary objective of this thesis is to identify a syllable based speech unit that can be used in concatenative speech synthesis for Indian languages.

Manual segmentation and labeling of speech units is tedious, time consuming and error prone. Moreover, due to variability in human perception, large inconsistencies are observed. It is hence necessary to automate identification and segmentation of appropriate speech units. Another objective of this thesis is to explore a signal processing technique to identify speech units suitable for speech synthesis and to automatically extract them from continuous speech.

In this thesis, the extraction of syllable units from the speech data has been automated by using a group delay based segmentation algorithm [23] (described in detail in Chapter 4). Speech units generated by this method are not exact syllables in the linguistic sense, but "syllable-like" units having a vowel nucleus with optional preceding and/or following consonants. These speech units are used to build unit selection synthesizers that produce speech with a high degree of naturalness. In contrast to diphone-based synthesizers that need elaborate prosody rules to produce natural speech, this approach does not need such extensive prosody rules. Instead, it relies on the F_0 based component of the target cost for selecting the best unit required (described in detail in Chapter 3). “Syllable-like” units identified using a group delay based segmentation algorithm are used with the Festival voice building framework [7, 24] to build a cluster unit selection synthesizer (described in detail in Chapter 5) for Tamil and Hindi.

Unit selection based synthesizers, however, run only on high end machines with RAM requirements ranging from 200 MB to 500 MB and hard disk requirements ranging from 500 MB to 800 MB, disallowing these synthesizers from being used on low end machines or embedded devices. The simplest choice to port these large synthesizers on embedded devices by reducing the repository sizes, is not a good option as it reduces the number of available speech units and prevents unrestricted speech synthesis. This approach can however be used for limited text-to-speech applications. In this thesis, distributed speech synthesis (DSS) [25] is proposed for unrestricted speech synthesis on embedded devices with limited memory and computational capabilities. The DSS is a client-server approach to speech synthesis. In this approach, low memory and computational tasks are done on the embedded device or client end, and the memory and computational intensive tasks are done on a server. The waveform repositories and the lexicon are also kept at the server end. This approach introduces an intermediate component, a communication network, to transfer intermediate data between the server and the client. The architectural design and implementation mechanism for a DSS system using Festival, a large speech synthesizer framework and Flite, a small memory footprint synthesizer, have been developed in this thesis. Taking mobile phones as a typical example of embedded devices requiring a speech synthesis interface, the DSS client has been implemented in Symbian C++ [26], a programming language for mobile phones (described in detail in Chapter 6).

1.5 Organization of the thesis

The rest of this thesis is organized as follows: Chapter 2 discusses the importance of prosodic modeling in speech synthesis and presents the techniques currently used for the same. This chapter introduces unit selection synthesis as an alternative when prosody models are not available. Chapter 3 describes the Festival speech synthesis framework used in the Indian language speech synthesis system developed. The emphasis is on the two different speech synthesis techniques that the framework supports. Chapter 4 outlines the group delay based segmentation algorithm used to generate the speech units. Chapter 5 explains how the segmentation algorithm is used to create various types of speech units, namely, monosyllables, bisyllables and trisyllables, and how the generated speech units are integrated into the Festival speech synthesis framework. An evaluation of the TTS system using Mean Opinion Scores (MOS) [27] on a five point scale is also presented in this chapter. In Chapter 6, speech synthesis for embedded devices using distributed speech synthesis is discussed. Chapter 7 gives a summary of the thesis and presents the directions for future work.

1.6 Major contributions of the thesis

The major contributions of the work presented in this thesis are as follows:

- Identification of a new syllable based speech unit suitable for concatenative speech synthesis,

- Demonstration of the usage of an automatic segmentation algorithm for generating these speech units,
- Development of natural sounding text-to-speech systems for Tamil and Hindi using the proposed units,
- Investigation of techniques to develop distributed speech synthesis systems for embedded devices.

CHAPTER 2

Approaches to Text-to-Speech Synthesis

2.1 Introduction

In this chapter we take a close look at two important issues in text-to-speech synthesis, namely, prosody modeling and waveform generation, and present a review of popular techniques for the same. These two steps are important for generation of *natural* sounding speech. At the perceptual level, naturalness in speech is attributed to certain properties of the speech signal related to audible changes in pitch, loudness and syllabic length, collectively called prosody. Acoustically, these changes correspond to the variations in the fundamental frequency(F_0), amplitude and duration of speech units [2, 4]. Prosody is important for speech synthesis because it conveys aspects of meaning and structure that are not implicit in the segmental content of utterances. Another interesting aspect of prosody is that it operates on longer linguistic units than the basic speech units and hence is referred to as a suprasegmental feature of speech. Generating the right prosody for synthetic speech in TTS systems is a difficult task because the input text mostly contains little or no explicit information about the desired prosody for the text and it is extremely difficult to deduce prosodic information automatically.

As explained in the previous chapter, the output of the text analysis module is a phonetic or linguistic transcription of the input text along with the desired pitch, duration and intensity information of the output waveform. This specification is often called a *target specification*. The DSP module is expected to produce synthetic speech close to the target specification. First generation speech synthesizers (Klatt-Talk, DEC-Talk) used formant synthesis techniques to do this [6]. Even though these synthesizers are capable of unlimited speech synthesis, the output does not sound natural. With synthetic speech being produced using models of the vocal tract system, synthesis of several voices (male, female, young, old etc) was possible. However, this approach requires a complex and coherent set of parameters every 2-10 milliseconds. Since finding the right set of parameters and their values for this technique is difficult, this technique does not produce the desired results. An alternate solution that is currently being pursued is called time-domain concatenative speech synthesis. This method primarily involves storing a set of speech segments (synthesis units), retrieving appropriate units using a selection criteria and concatenating chosen segments for synthesis. Waveform modification algorithms are usually applied to smoothen the segment transitions and to match the specified prosodic characteristics. This technique is capable of producing natural sounding synthetic speech with personal characteristics of specific speakers [28].

2.2 Approaches to prosody modeling

The task of modeling prosody is often subdivided into modeling the following constituents of prosody - phrasing, duration, intonation and intensity. Two major approaches for prosody modeling are the rule based approach and the corpus based approach. In the rule based approach, linguistic experts derive a complicated set of rules to model prosodic variations by observing natural speech. In the corpus based approach, a well-designed speech corpus, annotated with various levels of prosodic information is used. The corpus is analyzed automatically to create prosodic models which are then evaluated on test data. Based on the performance on test data, the models are then improved.

2.2.1 Prosodic phrasing

Sentences generally exhibit some form of prosodic structure with some words in a given sentence tending to group naturally while others introduce breaks. As an example, in the phrase *"I feel great about it, we should celebrate!"* there are two main prosodic phrases with the boundary at the punctuation mark. Prosodic phrasing involves finding these types of meaningful prosodic phrases, which may or may not be explicit. Prosodic phrasing is important because it not only increases the understandability of synthesized speech but also helps ascribe meaning to certain parts of the synthesized speech, in the same way humans do, by varying prosody. This is done by creating prosodic boundaries at explicit identifiers like punctuation marks, or at certain lexical or grammatical words, known to be phrase delimiters for a language. Several researchers have tried using variants of context-free grammars like augmented transition networks (ATNs), definite-clause grammars (DCGs) and unification grammars (UGs) to model syntactic-prosodic structures in languages and use them to identify the prosodic phrases in the input text [29–31]. Another approach is to use statistical models, with probabilistic predictors like CART decision trees, to predict prosodic phrases based on features such as the parts of speech of the surrounding words, the length of an utterance in number of words, the distance of a potential boundary from the beginning or the end of an utterance and whether surrounding words are accented [32].

Even though the rules based on punctuation are good predictors of prosodic phrases, there are many cases where explicit punctuation marks are not present to indicate the phrase boundaries. This problem is prominent in the case of Indian languages where there is little or no use of punctuation marks. Sridhar [33] uses an elementary deterministic rule based phrasing model for Hindi using the content/function word distinction.

2.2.2 Pitch modeling

Once the prosodic boundaries are identified, the speech synthesizer applies the prosody elements namely, duration, intonation and intensity, on each of the phrases and on the sentence as a whole. The primary factors that contribute to the intonation are the context of words and the intended meaning of sentences. Jurafsky [4] explains this with the following example. Consider the utterance *"oh, really"*. Without varying the phrasing or stress, it is still possible to have many variants of this by varying the intonation. For example, we might have an excited version *"oh, really!"* (in the context of a reply to a statement that one has won the lottery), a skeptical version *"oh, really?"* in the context of not being sure whether the speaker is being honest, or to an angry *"oh, really!"* indicating displeasure. Different pitch contours can, in fact, be observed for each of the different classes of sentences namely declarative, interrogative, imperative and exclamatory. Intonation is also influenced by the gender, physical state, emotional state and attitude of the speaker.

There are two approaches for automatic generation of pitch patterns for synthetic speech. The superpositional approach considers an F_0 contour as consisting of two or more superimposed components [34]. In this approach, the generated F_0 contour is the sum of a global component that represents the intonation of the whole utterance and the local components that model the change of F_0 over the accented syllables. The second approach, called as a linear approach considers an F_0 contour as a linear succession of tones. An example of the linear approach to pitch modeling is the Pierrehumbert or ToBI model that describes a pitch contour in terms of the pitch accents [35]. Pitch accents occur at stressed syllables and form characteristic patterns in the pitch contour. The ToBI model for English uses five pitch accents obtained by combining two simple tones, high (H) and low (L) in different ways. The model uses a H+L pattern to indicate a fall, a L+H pattern to describes a rise and an asterisk (*) to indicate which tone falls on a stressed syllable. The five pitch accents are as follows: H*, L*, L+H*, L*+H, H+L*.

2.2.3 Duration modeling

It has been observed that the duration of speech units in continuous speech can sometimes become as short as half their duration when spoken in isolation. Duration of a speech unit depends on several factors such as characteristics peculiar to the speech unit, influence of adjacent units and the number of speech units. The duration can also be a function of the sentence context. Duration modeling is important because TTS systems need to generate speech units with appropriate durations in order to produce natural sounding synthetic speech.

Several methods have been reported for duration modeling. In one approach (O'Shaughnessy [36], Bartkova and Sorin [37]), the intrinsic duration of a speech unit is modified by successively applying rules derived from analysis of speech data. Bartkova and Sorin [37] have analyzed several corpora to study speaker independent intrinsic durations and their modifications to come up with multiplicative rules and factors to modify an assigned baseline duration. In another approach large speech corpora are first analyzed by varying a number of possible control factors simultaneously to obtain duration models, such as an additive duration model by Kaiki [38], CARTs by Riley [3] and neural networks by Campbell [39]. The CARTs (classification and regression trees) proposed by Riley are data-driven models constructed automatically with the capability of self-configuration. The CART algorithm sorts instances in the learning data using binary yes/no questions about the attributes that the instances have. Starting at a root node, the CART algorithm builds a tree structure, selecting the best attribute and question to be asked at each node, in the process. The selection is based on what attribute and question will divide the learning data to give the best predictive value for classification. Riley uses a set of the following factors in building the tree: segment context (three segments to left, segment to predict, three segments to right), stress, lexical position (segment count from start and end of word, vowel count from start and end of word) and phrasing position (word count from start and end of phrase).

2.2.4 Intensity modeling

Of the four acoustic parameters that have been mentioned, intensity or loudness is often either neglected or modeled along with intonation primarily because it has been felt that intensity features are implied in intonation. Lee *et al.*, [40] uses artificial neural networks to predict syllable energy using factors like word position in a sentence, preceding and following pause duration, average pitch values, etc. Bagshaw [41] proposes an unsupervised model using a regression tree and associated rules for predicting energy at the phone level.

Even though prosody modeling has been investigated over the years, it is still difficult to automatically derived the prosody from text. Analysis of large amounts of speech data and expert knowledge are required for prosody modeling.

2.3 Concatenative speech synthesis

This section briefly reviews the approaches to concatenative speech synthesis and the signal processing techniques to modify pitch and duration of synthesized speech. Concatenative speech synthesis has become the most popular technique for waveform generation in TTS systems owing to its ability to produce natural sounding speech.

In concatenative speech synthesis, waveform is generated by selecting and concatenating the appropriate units from a database consisting of different types of speech units, namely, phones, diphones, syllables, words, phrases, recorded by a single speaker. The quality of synthesized speech is influenced by the quality of continuity of acoustic features (spectral envelope, amplitude, fundamental frequency and speaking rate) at the concatenation points and the availability of appropriate units with proper prosodic features in the database. If large units such as phrases or sentences are stored and used, the quality (intelligibility and naturalness) of synthesized speech is better, although the domain of synthesis does not become unrestricted text. If small units such as phonemes are used, an unrestricted text can be synthesized but the speech quality is largely degraded. Apart from the size of the speech unit, two major problems exist in concatenating the speech units to produce a sentence. With the duration of speech units in continuous speech being as short as half their duration when spoken in isolation, simple concatenation of speech units makes synthesized speech sound slow. The second problem is that the sentence stress pattern, rhythm and intonation are unnatural if speech units derived from the right contexts are not used. For example a */t/* before an */a/* sounds very different from a */t/* before an */s/* [4].

In order to resolve such problems, synthesis methods using speech units like a diphone are employed. Diphones provide a balance between context dependency and size (typically 1000-2000 units in a language). Even though the diphone based synthesizers produce reasonable quality speech, the prosody (pitch and duration) of each phone in the concatenated waveform does not correspond to the desired prosody. Several signal processing techniques have been developed for improving the prosody in concatenative speech synthesis. While signal processing and diphone concatenation can produce reasonable quality speech, the results are often not ideal - the primary reason being that a single example of each diphone or speech unit is not enough. Unit selection based concatenative synthesis is an attempt to address these issues by collecting several examples of each unit at different pitches, durations and linguistic situations, so that the unit is close to the target in the first place and hence requires less signal processing.

2.3.1 Pitch and Duration Modification

Even though diphone synthesizers produce a reasonable quality speech waveform, in many cases the pitch and duration of the speech units from database need to be modified to the pitch and duration required for proper sounding synthetic speech. Two popular signal processing techniques used for concatenative speech synthesis to modify pitch and duration of synthesized speech are the Time Domain Pitch Synchronous OverLap Add (TD-PSOLA) method [8] and the Harmonic plus Noise Model (HNM) method [9].

2.3.1.1 TD-PSOLA

In the PSOLA analysis-synthesis system, the speech signal is analyzed into a sequence of pitch-synchronous short-term signals (ST-signals). These analysis ST-signals are then modified, either in the time or in the spectral domain, in order to obtain a sequence of synthetic ST-signals, synchronized with a modified pitch contour. Finally, the synthetic speech is obtained by overlapped-addition of the synthetic ST-signal.

Pitch-synchronous overlap analysis

Short term signals $x_m(n)$ are obtained from digital speech waveform $x(n)$ by multiplying the signal by a sequence of pitch-synchronous analysis window $h_m(n)$: $x_m(n) = h_m(t_m - n)x(n)$ where m is an index for the short-time signal. The windows, which are usually Hanning type, are centered around the successive instants t_m, called pitch-marks. These marks are set at a pitch-synchronous rate on the voiced parts of the signal and at a constant rate on the unvoiced parts. The used window length is proportional to local pitch period. The unvoiced ST-signals are not converted to frequency domain since they will need only time scaling. For the voiced portions, a short term spectrum is computed by using DFT with time origin set to coincide with the pitch mark. It is then split into a global spectral envelope, a source component which is the short-term spectrum divided by the spectral envelope [42].

Frequency-domain modifications

Depending on the required prosodic modifications, the source or the envelope components are interpolated separately in order to obtain two separate rescalings of the frequency axis. Rescaling of the source component results in modification of the pitch. Changing the envelope component can be useful to modify the voice quality.

Time-scale modifications

Time-scale modifications are performed entirely in the time domain. The desired time scale modification can be defined as a time warping function mapping the analysis time scale onto the synthesis one. From the distribution of analysis pitch marks, a new sequence of synthesis pitch marks is generated such that the pitch contour is preserved.

Pitch-synchronous overlap synthesis

In the final step of this method, an ST-signal needs to be assigned to each synthesis pitch mark. Since a synthesis pitch mark generally will not correspond exactly to an analysis pitch mark, an approximation scheme which time averages the two nearest analysis ST-signals using an overlap add method, assigns a ST-signal to every new synthesis pitch mark. Manipulation of fundamental frequency is achieved by changing the time intervals between pitch markers. The modification of duration is achieved by either repeating or omitting speech segments.

2.3.1.2 Harmonic plus Noise Model

Harmonic plus Noise Model (HNM) is a pitch-synchronous analysis-synthesis system based on a harmonic plus noise representation of the speech signal. A prominent feature is that it does not require pitch marks to be determined as necessary in PSOLA-based methods. The harmonic part accounts for the quasi-periodic component of the speech signal; the noise part models its non-periodic components, which include friction noise and period-to-period variations of the glottal excitation [42].

The spectrum is divided into two bands. The time-varying maximum voiced frequency determines the limit between the two bands. During unvoiced frames the maximum voiced frequency is set to zero. In the lower band, the signal is represented solely by harmonically related sine waves with slowly varying amplitudes, and frequencies. Here, $h(t) = \sum_{k=1}^{K(t)} A_k(t) \cos(k\theta(t) + \phi_k(t))$ where $\theta(t) = \int_{-\infty}^{t} \omega_0(l)dl$, $A_k(t)$ and

$\phi_k(t)$ are the amplitude and phase at time t of the kth harmonic, $\omega_0(t)$ is the fundamental frequency and $K(t)$ is the time-varying number of harmonics included in the harmonic part. The upper band, that contains the noise part is modeled by an auto regression model and it is modulated by a time-domain amplitude envelope. The noise part, $n(t)$, is therefore supposed to have been obtained by filtering a white Gaussian noise $b(t)$ by a time-varying, normalized all-pole filer $h(\tau, t)$ and multiplying the result by an energy envelope function $w(t)$, such that $n(t) = w(t)[h(\tau, t) * b(t)]$

The first step of HNM analysis consists of estimating pitch and maximum voiced frequency. Using the stream of the estimated pitch values, the position of the analysis instants are set at a pitch-synchronous rate (regardless of the exact position of glottal closure). In voiced frames, the amplitudes and phases of the sinusoids composing the harmonic part are estimated by minimizing a weighted time-domain least-squares criterion. This time-domain technique combined with the relatively short duration of the analysis frame in the voiced parts of the signal (two pitch periods) provides a very good match between the estimated harmonic part and the original speech signal. The noise part is modeled by an all-pole filter estimated from 40ms of signal located around the center of the analysis frame.

Synthesis is also performed in a pitch-synchronous way using an overlap and add process. The harmonic part is synthesized directly in the time-domain as a sum of harmonics. The noise part is obtained by filtering a unit-variance white Gaussian noise through a normalized all-pole filter. If the frame is voiced, the noise part is filtered by a high-pass filter with cutoff frequency equal to the maximum voiced frequency. Then, it is modulated by a time-domain envelope synchronized with the pitch period. This modulation of the noise part is shown to be necessary in order to preserve the naturalness of some speech sounds, such as voiced fricatives.

2.3.2 Unit selection synthesis

Even though speech synthesis by concatenation of subword units like diphones produces clear speech, it does not have naturalness mainly because each diphone has only a single example. First of all, signal processing inevitably incurs distortion, and the quality of speech gets worse when the pitch and duration are stretched by large amounts. Furthermore, there are many other subtle effects which are outside the scope of most signal processing algorithms. For instance, the amount of vocal effort decreases over time as the utterance is spoken, producing weaker speech at the end of the utterance. If diphones are taken from near the start of an utterance, they will sound unnatural in phrase-final positions.

Unit selection synthesis is an attempt to address this problem by using a large database of speech with a variable number of units from a particular class. The goal of this method is to select the best sequence of units from all the possibilities in the database, and concatenate them to produce the final speech. By selecting units closest to the target, the extent of signal processing required to produce prosodic characteristics are reduced and thus minimize distortion of the natural waveforms. The unit selection is based on two cost functions. The target cost, $C^t(u_i, t_i)$, is an estimate of the difference between a database unit, u_i, and the target, t_i, which it is supposed to represent. The concatenation cost, $C^c(u_{i-1}, u_i)$, is an estimate of the quality of a join between consecutive units (u_{i-1}) and (u_i). The unit that minimizes both costs is selected. In this section we outline two unit selection techniques used in different speech synthesis systems [10].

2.3.2.1 Unit selection used in the CHATR synthesis system

The first stages of synthesis of the CHATR system [10] transforms the input text into a target specification, which is essentially the string of phonemes required to synthesize the text, annotated with prosodic features (pitch, duration and power). The speech database containing the candidate units can be viewed as a state transition network with each unit in the database being represented by a separate state. Because any unit can potentially be followed by any other, the network is fully connected. The task of picking the best set of units is performed using the Viterbi algorithm in a similar way to HMM speech recognition. Here the target cost is the observation probability and the concatenation cost is the transition probability.

2.3.2.2 Unit selection used in the Festival synthesis system

The Festival [24] synthesis system uses a cluster unit selection technique for selecting speech units from a speech database. In this method, the speech inventory is divided into clusters, where each cluster holds units of the same phone class based on their phonetic and prosodic context. The appropriate cluster is selected for a target unit, offering a small set of candidate units. This process is synonymous to finding the units with lowest target cost as described in the previous section. An optimal path is then found through the candidate units based on their distance from the cluster center and an acoustically based join cost [11].

By using a much larger database which contains many examples of each unit, the unit selection synthesis often produces more natural speech than the diphone synthesis. Some systems then use signal processing to make sure the prosody matches the target, while others simply concatenate the units.

2.4 Summary

In this chapter we have reviewed various approaches to prosody modeling. These techniques can primarily be divided into rule-based approaches or data-driven approaches. For prosody modeling in Indian languages, neither of these techniques have been extensive pursued. The primary reason for this being the lack of proper and complete annotated databases for Indian languages. Creating properly annotated databases is a time consuming effort. It is in this context that we have attempted to build a TTS systems that requires minimal prosody modeling. It is evident that this is possible if we are able to find a speech unit that intrinsically has sufficient prosody features and can be used with a unit selection technique. Nowadays it is affordable to have speech repositories with a large number of possible speech units of a language in different prosodic contexts.

The Festival speech synthesis framework allows the development of such a system using its incorporated unit selection algorithm. In this thesis, we work on a new speech unit and integrate it with the Festival system. In the next chapter, we describe the Festival speech system framework that enables inclusion of the new speech unit.

CHAPTER 3

The Festival Text-to-Speech System

3.1 Introduction

This chapter presents an overview of the Festival TTS framework [24] and the unit selection algorithm used to build a TTS system for an Indian language. Festival is a free, language independent speech synthesis engine that can be used for developing text-to-speech synthesis systems. It is highly flexible and has a modular architecture. Each phase of synthesis, as introduced in Chapter 1, is developed as a module here. It includes modules for text processing, linguistic/prosodic processing and waveform generation. Any of these modules can be rewritten to incorporate new techniques. In this thesis, we focus on a new automatically generated, syllable based unit. The TTS system for an Indian language is built using these speech units and evaluated. The primary advantage of this speech unit is that, when used with Festival's unit selection procedure, natural sounding synthetic speech could be generated without any extensive prosodic modeling. This technique can be used to develop TTS systems for Indian languages without the need to analyze large amounts of speech data and develop extensive prosody models. In this chapter, we describe two synthesis techniques supported by the Festival framework namely, diphone synthesis and unit selection synthesis. Festival's architecture for these two techniques is also explained.

3.2 Diphone synthesis

In this section, we review the diphone synthesis technique in Festival with reference to a speech synthesizer for Indian languages developed using this technique by Sridhar and Murthy [33]. Diphones provide a trade-off between capturing co-articulation effects, minimizing discontinuities at concatenation points and being relatively small in number. A diphone is made of two connected half phones and captures the transition between two phones by starting in the middle of the first phone and ending in the middle of the second one. In diphone synthesis, with only one instance of the speech unit being available, extensive prosodic modifications have to be applied to obtain good quality speech.

The first task involved in diphone synthesis is to identify the set of diphones that will cover the language for which the synthesizer is being built. In [33], Sridhar and Murthy have identified 1702 diphones to cover 4 Indian languages, Telugu, Hindi, Kannada and Tamil. An inventory of the diphones also includes information such as whether the diphone contains a vowel or consonant, the place of articulation for consonants and vowel length. The identified diphones are embedded in carrier words to ensure that they are pronounced clearly and consistently. Utterances of carrier words are then recorded and labeled.

In [33], Sridhar has also done extensive studies on prosody modeling for Indian languages. A classification and regression tree (CART) based model is used for phrase break prediction in Telugu. For Hindi, a deterministic, rule based phrasing model that uses punctuation and content/function word distinction is built. This model uses distance information for phrase break prediction. They have also proposed a feature called 'morpheme tag' for use in the phrasing models for Telugu. The classification and regression tree (CART) based duration models are used for segmental duration prediction for Telugu and Hindi. A rule-based intonation model is also used for applying global properties of intonation over prosodic phrases and local fall-rise patterns at the syllable level. In diphone based synthesis, it is required to use signal processing techniques to modify duration and pitch. Festival uses residual excited linear predictive coding (LPC) based re-synthesis [43]. These algorithms for modification are pitch-synchronous techniques and hence require information about pitch periods. Pitchmarks are extracted using the 'pichmark' program [44] in Edinburgh speech tools. After the pitchmarks have been obtained, pitch-synchronous LPC parameters and residuals are extracted using 'sig2fv' and 'sigfilter' programs [44] in Edinburgh speech tools. The diphone database is finally coded into the formats required by the waveform synthesizers, the UniSyn synthesizer [24] and the OGI diphone synthesizer [45] available with Festival.

TTS systems based on diphone synthesis need prosodic models to produce good speech output. The prosodic analysis for these models require a database of speech annotated with linguistic and prosodic labels. Tools are also required to generate appropriate linguistic information essential to predict prosody from text. With work on these aspects still in its infancy for Indian languages, there is a need to find techniques to build synthesizers that do not rely on prosodic models. It is in this context that we examine the unit selection synthesis techniques.

3.3 Unit selection synthesis

As introduced in Chapter 2, the unit selection synthesis technique selects the best sequence of speech units from a database of speech units and concatenates them to produce speech. These selected speech units should satisfy the following two constraints.

They should best match the *target specification* given by the linguistic components of the text analysis module and 2) they must be the best units that join together smoothly when concatenated. The cost associated with the first constraint is called the *target cost* and the cost associated with the the second constraint is called the *concatenation or join cost.*

The *target specification* is a sequence of speech units along with features related to the phonetic and prosodic context for each unit. The phonetic context features include the identity of a particular speech unit, position of the speech unit in the word and the phonetic features of the previous and following speech units. The prosodic context features include the pitch, duration and stress of the particular unit and the prosodic features of the preceding and following units. Similar information is associated with each unit of the speech database.

Festival uses a clustering technique [11] to organize the units in the speech database according to their phonetic and prosodic context. For example, if there is a speech unit */an/*, all instances of the speech unit */an/* with different phonetic and prosodic contexts belong to the same class. Each class is organized as a decision tree whose leaves are the various instances of the speech unit. The branches of the decision tree are questions based on the prosodic and phonetic features that describe the units. During synthesis time, for each target unit in the target list, its decision tree is identified from the speech database. Using the target specification for each unit and the decision tree, a set of candidate units that best match the target specification is obtained.

For clustering, Festival defines an acoustic measure of distance between two units of the same type. The acoustic vector for a frame includes mel-frequency cepstral coefficients, fundamental frequency F_0, energy and delta cepstral coefficients. This acoustic measure of distances is used to define the *impurity* of a cluster of units as the mean distance between all members. The CART method is used to build the decision tree such that the branches correspond to questions that minimize the impurity of the sub-clusters. With the whole process of clustering the speech database into clusters done offline, the only task that is done during synthesis time is to use the decision tree to find a set of possible candidate units. To join the consecutive candidate units from clusters selected by the decision trees, Festival uses a join cost which uses a frame based Euclidean distance. The frame information includes F_0, mel-frequency cepstral coefficients, energy and delta cepstral coefficients.

In the synthesis phase, the text processing module of Festival first generates a target specification for the input text. For each target, based on questions from the target specification, the CART for that unit type gives the appropriate cluster which provides a set of candidate units. A function *Tdist(U)* is defined as the distance of a unit U to its cluster center (synonymous to finding the target cost). Another function *Jdist*(U_i,U_{i-1}) is also defined to find the join cost between a candidate unit U_i and the previous candidate unit U_{i-1}. A Viterbi search is then used to find the optimal path through the candidate units that minimizes the following expression: $\sum_{i=1}^{N}(Tdist(U_i) + W * Jdist(U_i, U_{i-1}))$ where W is a weight that can be set to give more importance for the join cost over target cost.

3.4 Festival TTS system

Having introduced the two important synthesis techniques available with Festival, we briefly describe the architecture of the Festival speech synthesis framework [24]. Figure 3.1 shows the block diagram of the diphone synthesis based TTS system in Festival. Prosody modeling is done in the prosodic phrasing, segmental duration generation and

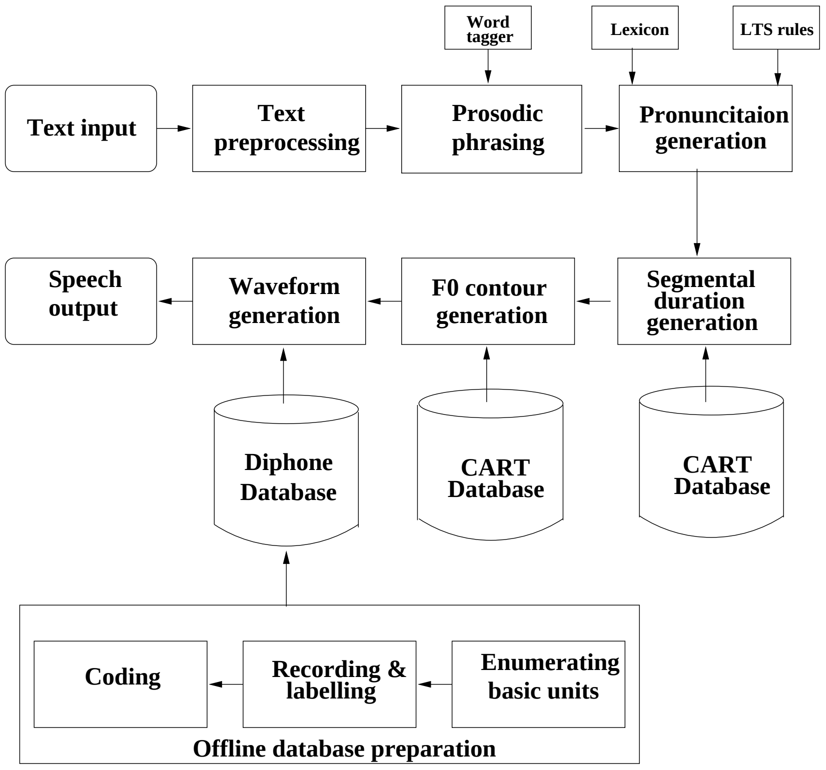

Fig. 3.1: Block diagram of TTS system using diphone synthesis in the Festival speech synthesis framework

F_0 contour generation modules. Units selected from the database are modified based on the specifications from these prosodic models. In the unit selection based synthesis technique, the prosodic modules are replaced by the unit selection module and CART based cluster database as shown in Figure 3.2.

The input to the TTS system is the transliteration of a text in an Indian language. The prosodic phraser used to chunk text is based on punctuation and distance information from the previous break and the distance to the next punctuation. The pronunciation generation module generates the sequence of basic units using a lexicon of units and letter-to-sound rules. The lexicon is a list of all speech units - monosyllables, bisyllables and trisyllables, present in the waveform repository. The letter-to-sound rules are framed in such a way that each word is split into its largest constituent syllable

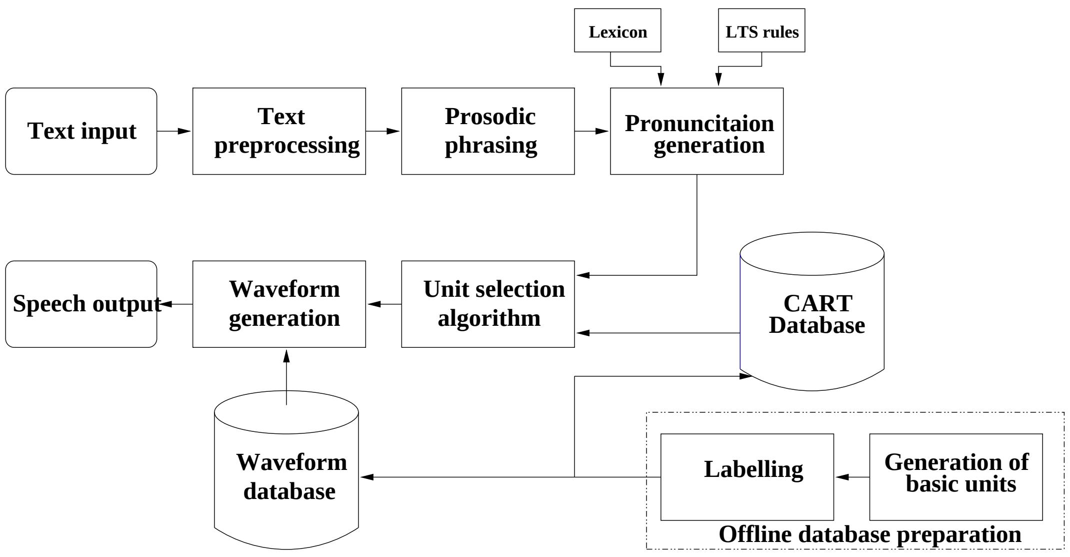

Fig. 3.2: Block diagram of TTS system using unit selection based synthesis in the Festival speech synthesis framework

units: trisyllables first, bisyllables only if trisyllable combinations are not present, and finally monosyllables if bisyllable combinations are also not present. As the pronunciation of most of the words in Indian languages can be predicted from their orthography, these rules suffice to generate correct pronunciations. The unit selection algorithm module generates a target specification for the speech units that have been identified and picks the best sequence of speech units that minimizes both the target cost and the join cost. The waveforms of these speech units are then concatenated to produce synthetic speech.

3.5 Summary

In this chapter we have reviewed the various speech synthesis techniques available with Festival. In the unit selection technique, the quality of synthetic speech is a function of the available units present in the database. For good quality synthesis, all the units of the language should be present. Moreover, the units should also be generic so that they can be used for unrestricted text-to-speech synthesis. In diphone synthesis, the quality of synthetic speech is dependent on the prosodic rules that are present. If appropriate rules are present, proper prosodic specifications can be generated. In the case of Indian languages, the primary reason for poor quality of speech synthesized using diphone synthesis is the lack of proper prosodic rules. This is in turn because of the unavailability of proper databases annotated with prosodic information like ToBI indices [35] for English. In this thesis we attempt to compensate for this by identifying a speech unit that intrinsically has enough prosodic and acoustic information within it. The unit selection based speech synthesis is used to choose units from a database populated with these speech units in various prosodic contexts. The second issue that we address is the generation of these units. If manually done, this task becomes tedious, time consuming and error prone. Moreover due to variability in human perception, there may be inconsistencies in manual segmentation. In this thesis we address this issue by using an automatic segmentation algorithm. The next chapter describes the automatic segmentation algorithm.

CHAPTER 4

Automatic Generation of Speech Units

4.1 Introduction

As illustrated in Chapter 1, syllables are a better choice for speech synthesis in Indian languages. However, manually identifying, segmenting and labeling syllable units from speech data for a particular language is tedious, time consuming and error prone. Moreover, due to variability in human perception, large inconsistencies are observed. It is hence of great advantage to automate identification and segmentation of syllable units. In this thesis, an existing group delay based algorithm is used to automatically extract syllable units from continuous speech. In this chapter the group delay based algorithm and its important underlying signal processing concepts are reviewed.

4.2 Segmentation using minimum phase group delay functions

The Short Term Energy(STE) of a speech signal is a simple time domain method of processing and characterizing important features of a speech signal, especially variations in amplitude. These amplitude variations are profound in regions of voiced and unvoiced speech. Syllables, typically of the form C^*VC^* (C:consonant, V: vowel), can be thought of as having a vowel-like nucleus with optional preceding and/or following consonants. In terms of the STE function, this is characterized by regions with high energy at the center and energy tapering at both ends. In the group delay based method for segmentation proposed by Prasad and Murthy [23], the short term energy alone is considered for segmentation of the speech signal. The STE function is characterized by minima and maxima along with local variations. The minima correspond to boundaries of syllable-like units. The group delay based segmentation algorithm smoothens the STE by eliminating the local variations and retaining the minima. The minima in the group delay function correspond to minima in the short term energy and are considered as segmentation points. The following sections briefly describe the important signal processing concepts used by the segmentation algorithm.

4.2.1 Group delay function and its properties

The negative derivative of the Fourier transform phase is defined as *group delay.* If the phase spectrum $\theta(\omega)$ of a signal is a continuous function of ω, the group delay function is defined as $\tau(\omega) = -\frac{d(\theta(\omega))}{d\omega}$. The value of the group delay function indicates the degree of nonlinearity of the phase and is expressed in seconds or samples. The advantage of the group delay function is that the additive property of the phase spectrum is retained in the group delay function. Another advantage is that the group delay function can be computed directly from the time-domain signal $x(n)$ without having to compute the unwrapped phase using the formula: $\tau(\omega) = Re\left[\frac{FT(x(n))}{FT(nx(n))}\right]$ where FT stands for the Fourier transform and Re stands for the real part. Two important properties of the group delay function are its additive and high resolution properties as illustrated in Figure 4.1 [23]. Here, three different systems have been taken into consideration: a system with one pole-pair at $(0.8, \pi/9)$ and $(0.8, 17\pi/9)$ (Figure 4.1 (a)), a second system with one pole-pair at $(0.8, \pi/4)$ and $(0.8, 7\pi/4)$ (Figure 4.1 (d)) and a third system with one pole-pair at $(0.8, \pi/9)$ and $(0.8, 17\pi/9)$ and another pole pair at $(0.8, \pi/4)$ and $(0.8, 7\pi/4)$ (Figure 4.1 (g)). Figures 4.1 (b) and 4.1 (e) are the magnitude spectra, and Figures 4.1 (c) and 4.1 (f) are the group delay spectra of the systems given in Figures 4.1 (a) and 4.1 (d) respectively. Since in the third system (Figures 4.1 (c)), both the poles of the first (Figures 4.1 (a)) and second (Figures 4.1 (d)) systems are combined together, the corresponding magnitude spectrum is a multiplication of the magnitude spectra given in Figures 4.1 (b) and 4.1 (e) as shown in Figures 4.1 (h). But the group delay spectrum is an addition of the group delay spectra shown in Figures 4.1 (c) and 4.1 (f), as evident in Figures 4.1 (i). From Figures 4.1 (h) and 4.1 (i), it is also clear that the group delay spectrum also ensures the high resolution property.

To demonstrate the high resolution property of the group delay spectrum, it has been compared with the magnitude and linear prediction (LP) spectra of the same signal as shown in Figure 4.2, taken from [23]. Figure 4.2(a) shows the z-plane plot of the system that consists of two complex conjugate pole pairs. Figure 4.2 (b) is the corresponding magnitude spectrum. Figure 4.2 (c) shows the spectrum derived using LPC analysis and Figure 4.2 (d) is the corresponding group delay spectrum. It is evident that the two peaks are resolved much better in the group delay spectrum.

The group delay function derived from a minimum phase signal [1] is called *minimum*

[1]A discrete time signal $x(n)$ is called a *minimum phase* signal if both $x(n)$ and its inverse $x^i(n)$ are energy bounded and one-sided signals. In terms of poles and zeros of the z transform, $x(n)$ is called minimum phase if and only if all the poles and zeros of the transform lie within the unit circle. Let the Fourier transform of minimum phase signal $x(n)$ be represented by $X(e^{j\omega}) = |X(e^{j\omega})|e^{j\theta(\omega)}$ where $|X(e^{j\omega})|$ and $e^{j\theta(\omega)}$ are the magnitude and phase spectra respectively. The magnitude spectrum and unwrapped phase function of the signal can then be represented as $ln|X(e^{j\omega})| = \frac{c(0)}{2} + \sum_{n=1}^{\infty} c(n)cos(n\omega)$ and $\theta(\omega) = -\sum_{n=1}^{\infty} c(n)sin(n\omega)$ respectively where $c(n)$ are cepstral coefficients.

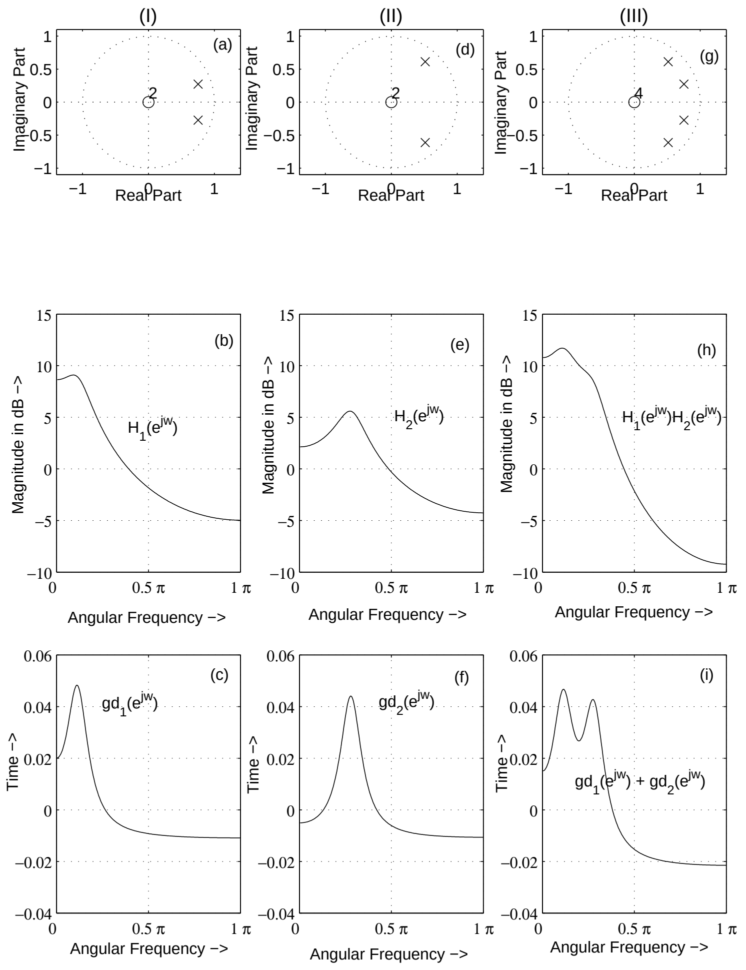

Fig. 4.1: Resolving power of group delay spectrum: the z-plane, magnitude spectrum and group delay of the cases when I) a pole–pair at $(0.8, \pi/9)$ and $(0.8, 17\pi/9)$ inside the unit circle, II) a pole–pair at $(0.8, \pi/4)$ and $(0.8, 7\pi/4)$ inside the unit circle, and III) a pole-pair at $(0.8, \pi/9)$ and $(0.8, 17\pi/9)$ and another pole pair at $(0.8, \pi/4)$ and $(0.8, 7\pi/4)$, inside the unit circle.

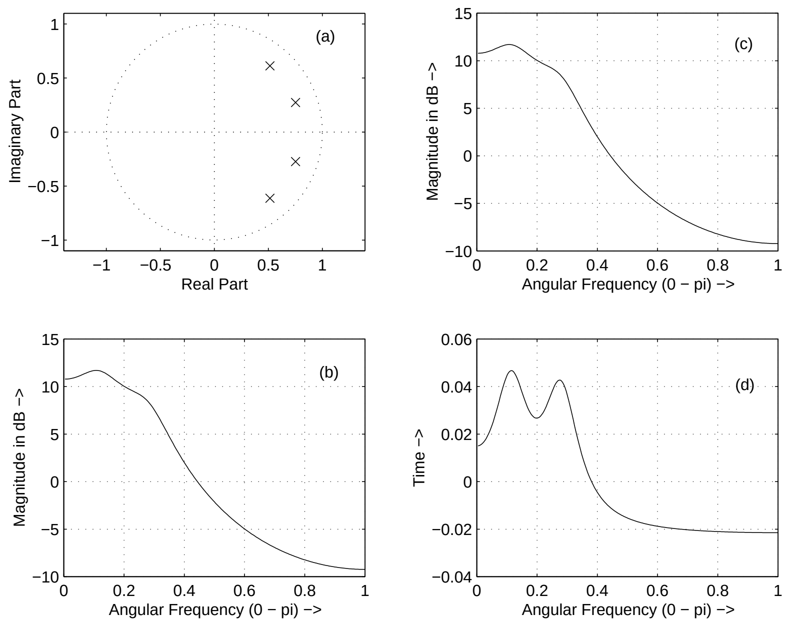

Fig. 4.2: Comparison of the minimum phase group delay function with linear prediction (LP) spectrum. a) The z-plane with two poles inside the unit circle, b) the magnitude spectrum of the system shown in (a), c) the LPC spectrum of the system shown in (a), d) the group delay spectrum of the system shown in (a)

phase group delay function. In the minimum phase group delay function, poles are characterized by peaks while zeros are characterized by valleys. The group delay functions of non-minimum phase signals do not possess this property [23].

4.2.2 Short-term energy function and its properties

The short-term energy (STE) of a signal corresponds to an estimation of the energy of a signal over a short window of N samples, and is defined as $E_n = \sum_{m=n-N+1}^{n} s^2(m)$. In other words, the short-term energy at sample n is the sum of the squares of N samples at $n - N + 1$ through n. The short-term energy function and the Fourier transform magnitude spectrum possess many similar properties. Both the STE function and the magnitude function are real valued, positive and non-zero in nature. However, while the magnitude spectrum of the short-term Fourier transform is even-symmetric about the y - *axis*, the STE function is not symmetric. To make the STE function resemble the magnitude spectrum, the STE function can be reflected about the y - *axis*. The concatenation of this reflected function with the original STE function appears like a magnitude spectrum, except for the units and ranges along the x - *axis* [23].

4.2.3 Root cepstrum approach to derive a minimum phase signal

A minimum phase signal can be derived from the magnitude spectrum of any type of energy bounded signal using the root cepstrum approach [46]. If $X(e^{j\omega})$ is the discrete Fourier transform of a signal $x(n)$, the root cepstrum is obtained by finding the inverse DFT of the magnitude spectrum raised to γ, where $0 < \gamma < 2$. It has been shown that the causal portion of the root cepstrum is a minimum phase signal [23].

4.2.4 The segmentation algorithm

The group delay based segmentation algorithm [23] is as follows:

1. For a given digitized speech signal, compute its corresponding short term energy (STE) sequence using overlapped windows.

2. Symmetrize the STE sequence. The symmetrized sequence is now viewed as an arbitrary magnitude spectrum.

3. Invert the sequence to reduce the dynamic range and prevent large peak excursions. Since the sequence is inverted in this step, it is the peaks of the group delay function, not the valleys, that now correspond to segmentation points.

4. Compute IDFT of the symmetrized sequence. The causal part of the resulting sequence is a minimum phase signal [46].

6. Compute the group delay function of the windowed causal sequence.

7. Locations of the positive peaks in the group delay function give approximate boundaries of syllable-like segments.

Segmenting the speech signal at the minimum energy points gives rise to units that have C^*VC^* structure (C: consonant, V: vowel). The group delay based algorithm is able to locate polysyllables by adjusting the so-called "window scale factor (WSF)". This is described in detail in the next chapter. As an example, consider the segmentation of the speech signal for an utterance of the Tamil phrase */ennudaya ThAimozhi/*. The speech signal waveform, its short term energy, and the group delay are shown in Figure 4.3. The positive peaks in the group delay function give the syllable boundaries, and are marked by the vertical lines. For this example, the boundaries are for the following units: */en/, /nud/, /day/, /ya/, /ThAim/, /moz/, /zhi/*, where (i)*/en/* is a VC unit, (ii) */nud/, /moz/* are CVC units and (iii) */ya/, /zhi/* are CV units. These units are assigned labels after listening to them.

It should be noted that the group delay based segmentation algorithm is capable of identifying boundaries at semivowel regions also, which are otherwise very difficult to obtain automatically. An example of this can be seen as in the case of segmentation of speech unit */ya/* from the word */ennudaya/* as shown above.

4.3 Summary

In this chapter the group delay based segmentation algorithm used for generating speech units for concatenative speech synthesis was discussed. This method is not

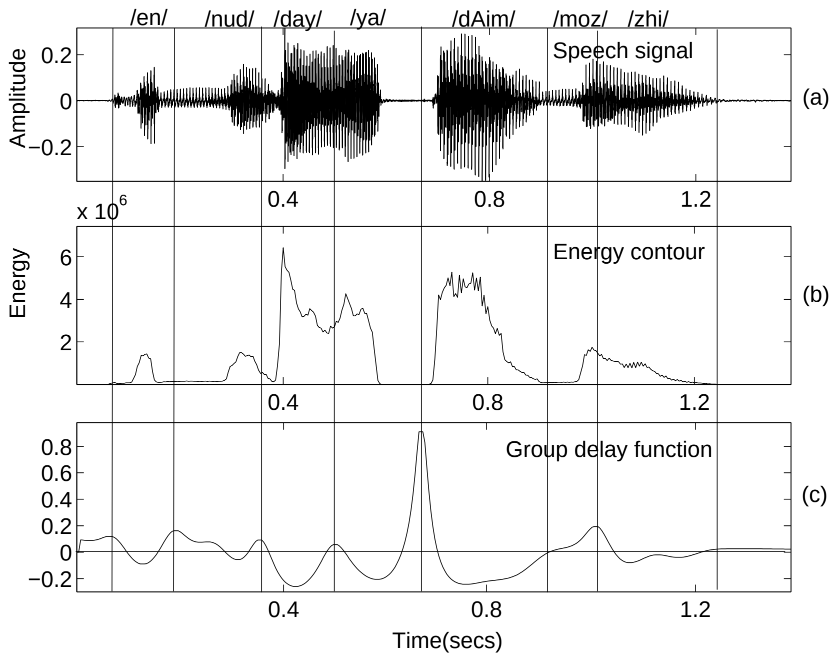

Fig. 4.3: Segmentation of the speech signal for the Tamil phrase *"ennudaya ThAimozhi"* (a) The speech waveform (b) Short term energy and (c) corresponding minimum-phase signal's group delay

only consistent but also produces results that are close to manual segmentation. In the next chapter, the steps required to generate speech units - monosyllables, bisyllables and trisyllables using the segmentation algorithm and how they are used to to build a synthesizer using the Festival framework are discussed.

CHAPTER 5

Unit Selection Synthesizers for Tamil and Hindi

5.1 Introduction

The group delay segmentation algorithm and the Festival framework discussed in the previous chapters form the basis for the unit selection synthesizers developed for Tamil and Hindi in this thesis. In this chapter the steps followed to build these synthesizers along with the evaluations of the speech units and the synthesizers are discussed. The results of the evaluation show the effectiveness of the speech units and the appropriateness of the synthesis technique.

5.2 Identifying syllable-like units

In the group delay based segmentation algorithm described in the previous chapter, the minimum phase group delay function was computed for a windowed causal sequence of the root cepstrum. The size of the cepstral lifter window is denoted as N_c [46, 47]. The frequency resolution in the magnitude spectrum as well as in the group delay spectrum depends on the size of the cepstral lifter N_c. N_c is defined as follows:

$$N_c = \frac{Length\ of\ the\ short\ term\ energy\ sequence}{WSF}$$

where the length of the STE sequence corresponds to the number of samples in the STE sequence and the window scale factor (WSF) represents a scaling factor used to truncate the cepstrum [48]. The window scale factor takes integer values.

If N_c is high, the resolution will also be high, that is, closely spaced boundaries will be better resolved. In other words, the syllable-like units generated will have a shorter duration and will correspond to monosyllables. On the other hand as N_c is made low, the resolution decreases and larger units corresponding to polysyllables - bisyllables and trisyllables, start appearing. Different types of syllable-like units required for speech synthesis are hence generated by changing the value of N_c as required. In this thesis, the use of units up to trisyllables has been explored.

The number of mono-, bi-, and trisyllables for 100 words taken from 30-sentences are given in Table 5.1 for different values of WSF. It is observed from Table 5.1 that with the WSF value being inversely proportional to N_c, for smaller values of the WSF value, monosyllables appear predominantly. As the value of the WSF increases, larger number of polysyllables are obtained. Figure 5.1 shows the syllable-like units

Table 5.1: Number of *syllable-like* units for different window scale factors WSF

WSF	Monosyllables	Bisyllables	Trisyllables
4	273	62	3
6	168	70	10
8	96	94	18
10	52	101	28

obtained from the speech signal waveform of the Tamil word */vanakkam/* for different values of the WSF using the segmentation algorithm. For higher values of the WSF,

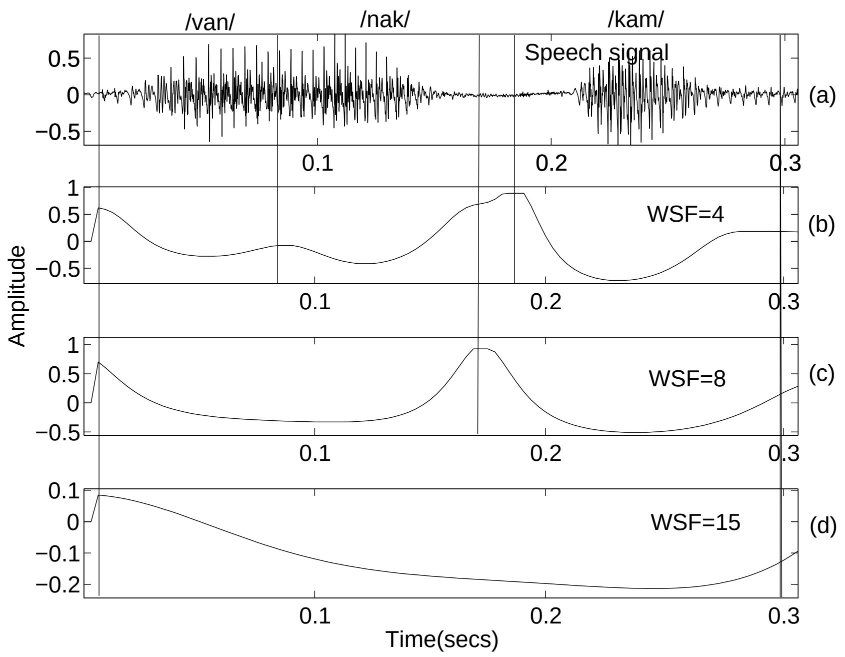

Fig. 5.1: (a) Speech signal waveform for the Tamil word */vanakkam/*. (b) For WSF = 4, boundaries of the monosyllables */van/*, */nak/*, */kam/* are identified (c) For WSF = 8, boundaries of one bisyllable */vanak/* and one monosyllable */kam/* are identified (d) For WSF = 15, boundary of trisyllable */vanakkam/* is identified

greater than 15, the segmentation algorithm gives word-like units. On varying the WSF factor from 15 to 25, the corresponding word boundary accuracy varies from 48.2% to 54.1%, with the maximum accuracy of 55.9% occuring for WSF = 22.

5.3 Improving the accuracy of syllable-like unit identification

The quality of synthetic speech generated using unit selection based synthesis is a function of the available units in the database. For good quality synthesis, all the units of the language must be present. Moreover, the units should be generic so that they can be used for unrestricted text-to-speech synthesis. The units should also be correctly labeled and should belong to one of the syllable groups - CV, VC or CVC or a combination of these groups. This facilitates the definition and use of well defined letter-to-sound rules during the pronunciation generation phase of the speech synthesizer. When the segmentation algorithm is run on continuous speech data corresponding to sentences, it is sometimes observed to give units that span across word boundaries. At times, short units that do not fall into syllable categories of CV, VC or CVC are also obtained. Since these units are unusable for speech synthesis, the following steps are performed to prevent the occurrence of such units:

- Word level segmentation is performed manually. The automatic segmentation algorithm is then run on a speech repository of words. This ensures that units that span across words do not appear.
- The speech repository of words is carefully listened to and correctly labeled with appropriate transliterated English text. A text segmentation algorithm is then run on each of the text labels to identify the syllables that are present in the corresponding speech data. For example, if the text segmentation algorithm was run on the text */vanakkam/*, it would identify three syllables - "*/van/*", "*/nak/*", "*/kam/*". This is done by identifying vowels in the text and combining them appropriately with adjacent consonants to form CV, VC or CVC units.

The number of syllable units identified by the text segmentation algorithm($|n_t|$) is then verified with the number of syllable-like units generated by the group delay based segmentation algorithm, ($|n_s|$). If the group delay based segmentation algorithm identifies more units than expected for each word ($|n_s| > |n_t|$), a set of syllable-like units with cardinality equal to the number of expected syllables($|n_t|$) is selected. This selection is done by choosing the first $|n_t|$ units from the list of $|n_s|$ units sorted in descending order based on segment duration, the assumption being that the units with longer duration would be suitable syllable-like units. The remaining ($|n_s - n_t|$) units are then merged appropriately with the ($|n_t|$) units. This heuristic ensures that no spurious units appear in the syllable database being created.

5.4 Creating a sample syllable database for speech unit evaluation

Since the entire procedure of enumerating syllable-like units and using them to create a unit selection synthesizer using the Festival framework was new, a sample syllable database was first created for evaluation. The sample syllable database was created using speech recordings of an existing news bulletin database, called DBIL [49]. The news bulletins were recorded by a native Tamil speaker in the laboratory environment. The news bulletins contain 6,421 sentences with an average of 6 words per sentence. The following steps were performed to create the sample database:

- Record the news bulletin sentences with a native Tamil speaker
- Perform manual word level segmentation of the recorded speech data

- Run the group delay based segmentation algorithm for different values of the WSF parameter on the word level speech data with the heuristics for improving the accuracy of units described above.
- Manually listen to the syllable-like units - monosyllables, bisyllables and trisyllables, generated by the previous step and assign appropriate transliterated labels for the sounds.

The above procedure for segmentation on the DBIL database resulted in 1,325 unique monosyllable-like units, 4,882 unique bisyllable-like units and 4,795 unique trisyllable-like units. The syllable-like speech units generated for the sample database were then integrated into the Festival framework to build a unit selection synthesizer for Tamil.

5.5 Building a unit selection synthesizer using the Festival framework

Once a speech repository is in place, the repository is integrated with the Festival framework. An outline of the steps to build a unit selection synthesizer are given below. A more detailed description of same is available in [7].

- Creating letter-to-sound rules and phoneset - A comprehensive set of letter-to-sound rules was created to syllabify the input text into the syllable-like units. These rules are framed in such a way that each word is split into its largest constituent syllable unit: trisyllables first, bisyllables only if trisyllable combinations are not present, and finally monosyllables if bisyllable combinations are also not present. The phoneset, which is a list of basic sound units for a particular language that the synthesizer supports, was created by enumerating all the speech units identified in the syllabification process.

- Building utterance structures for the database - Festival represents each speech unit internally with a data structure called an *'utterance'*. The *'utterance'* structure holds all the relevant phonetic and prosodic information related to a speech unit within this data structure. The phonetic information in an *'utterance'* structure describes the position of the speech unit in the word it appears and the information of units adjacent to it. Prosodic information holds information about the duration and pitch of the unit. Festival provides relevant scripts for building *'utterance'* structures for each speech unit.

- Generating speech unit clusters - As described in chapter 3, with Festival using a cluster based unit selection algorithm, a number of offline processes need to be done to create clusters of speech units belonging to the same class. These processes include building coefficients for acoustic distances (MFCC, F_0 and energy coefficients), creating distance tables for each class of units based on acoustic distances and generation of features for building CART trees. Festival provides scripts to perform these tasks. Once these are in place, Festival uses a CART building program - 'wagon' [44], to generate a CART tree for each class of units.

- Building the unit synthesizer - Using the letter-to-sound rules, phoneset and clusters of each speech unit built in the previous steps, Festival generates the necessary files that need to be used along with the core Festival speech synthesizer to build a unit selection synthesizer for a particular language using appropriate scripts.

5.6 Evaluation of the syllable-like units

The syllable-like speech units generated for the sample database were then integrated with the Festival framework to build a unit selection synthesizer for Tamil. As described in chapter 3, Festival uses a selection criteria with two costs, namely (a) target cost to determine how close the features of a speech unit features are to the desired, actual phonetic and prosodic features, and (b) concatenation cost that measures how well the speech units match and join with other speech units when concatenated. The unit that minimizes both the costs is then selected.

The sample speech synthesizer for Tamil is evaluated to verify whether the "syllable-like" speech unit is indeed a good candidate for use in concatenative speech synthesis. Based on the conclusions drawn from this evaluation high quality speech synthesizers are designed for Tamil and Hindi.

5.6.1 General evaluation of the syllable-like unit

In order to test the improvement of synthesized speech quality using syllable-like units, a perceptual evaluation of 20 sets of synthesized Tamil sentences was conducted using 20 native Tamil subjects. Each set had 4 sentences synthesized using different methods: the first sentence in each set was synthesized using a diphone synthesizer [33]; the second sentence was synthesized using monosyllables only; the third sentence was synthesized using both monosyllables and bisyllables units, with the monosyllables being used only when bisyllables are not present; the final sentence was created with trisyllables, bisyllables, and monosyllables. For the last case, each word used the largest possible syllable unit. As an illustration, the phrase */inda nikazchchiyil/*, contained following units for the 4 cases:

a) Diphones: */i-n/ /n-d/ /d-a/ /n-i/ /i-k/ /k-a/ /a-z/ /z-ch/ /ch-ch/ /ch-iy/ /iy-l/*

b) Monosyllables: */in/ /da/ /ni/ /kaz/ /chchi/ /yil/*

c) Mono- and bisyllables: */in/ /da/ /nikaz/ /chchiyil/*

d) Mono-, bi-, and trisyllables: */in/ /da/ /nikazchchi/ /yil/*

The subjects were asked to score the naturalness of each waveform on a scale from 1 to 5 (1=Bad, 2=Poor, 3=Fair, 4=Good 5=Excellent). The Mean Opinion Score(MOS) for each of the synthesis units is given in Table 5.2.

Table 5.2: MOS for sentences synthesized using different speech units

Diphone	Monosyllable	Bi- and monosyllable	Tri-, bi-, and monosyllable
1.34	1.47	3.74	3.97

The results of the first MOS test show that speech synthesis with syllable-like units is better than diphone based synthesis. The MOS for using only monosyllables is marginally better than that for diphone synthesis. Of the three different types of syllable-like units, trisyllable as the primary concatenation unit is preferred to the other two. The disadvantage is, of course, the number of such units needed. However, it is also evident from the scores that bisyllable units can also be used in place of trisyllables to achieve good results, but without significant increase in the number of units needed.

A second perceptual test was conducted to find which order of syllable-like units was best acceptable for synthesis. The combinations used were as follows:

a) Monosyllables at the beginning of a word and bisyllables at the end (order 1).

Example: */palkalaikkazaha/* is synthesized as */pal/ /kal/ /a/ /ik/ /kazaha/*.

b) Bisyllables at the beginning of a word and monosyllables at the end (order 2). Example: */palka/ /laikka/ /za/ /ha/*.

c) Monosyllables at the beginning and trisyllables at the end of a word (order 3). Example: */pal/ /kal/ /a/ /ikkazaha/*.

d) Trisyllables at the beginning and monosyllables at the end of a word (order 4). Example: */palkalaik/ / kaza/ /ha/*.

A set of 20 Tamil sentences are synthesized using each of these 4 combinations and the subjects were asked to score the naturalness with the MOS as in the first test. The MOS for different orders of speech units is given in Table 5.3. The results of the second test show that it is better to use large units (trisyllables or bisyllables) in the beginning, and monosyllables only at the end.

Table 5.3: MOS for sentences synthesized using different orders of syllable units

Mono- and bisyllable (order 1)	Bi- and monosyllable (order 2)	Mono- and trisyllable (order 3)	Tri- and monosyllable (order 4)
3.1	3.9	3.8	4.1

5.6.2 Prosodic evaluation of the syllable-like unit

Two perceptual tests were carried out to evaluate the extent of naturalness of synthetic speech generated by the speech synthesizers. The first prosodic evaluation test was to find how much of useful duration information was intrinsically present within the syllable-like units. The test was done by first synthesizing 100 sentences with an average of 6 words using the sample synthesizer. These sentences are then used to build a CART tree for duration modeling. A second CART tree was also generated using a single speaker news bulletin of 12 minutes duration and 121 read sentences. The CART trees built from both the synthesized and natural speech databases were then separately used as duration models for a diphone synthesizer developed in [33]. Three sets of 10 Tamil sentences each were synthesized using the different duration models and the subjects were asked to score the naturalness with the MOS as in the earlier tests. The first set had sentences synthesized with no duration modeling. The second set was synthesized with a CART duration model built using the single speaker news database. The sentences of the third set were synthesized with a CART duration model generated using synthetic speech from the sample synthesizer. The MOS for different sets is given in Table 5.4. This prosodic evaluation shows that the syllable-like unit intrinsically has sufficient duration information as it improves the quality of synthetic speech when used as a duration model. This test also indicates that synthesizers using the syllable-like unit as a speech unit might not require additional duration modeling.

A second perceptual evaluation test was carried out to evaluate the overall perceptual quality of the synthesized speech using the syllable-like unit. A passage of 15 sentences was synthesized and 20 subjects were asked to evaluate the overall perceptual quality based on the following aspects - a) intelligibility (whether the synthesized speech could be understood), b) naturalness (whether the synthesized speech sounded natural and not robotic in nature) and c) distortion (whether there were unnatural variations in prosody). The result of this test given in Table 5.5 shows that 70%-75% of

Table 5.4: MOS for Tamil sentences using duration modeling

Without duration modeling	With duration modeling learnt from synthetic speech	With duration modeling learnt from natural speech
1.1	2.5	3.1

the speakers felt the synthesized speech was natural sounding, intelligible and had low distortion. However some of the listeners indicated that there were unnatural changes in intensity and intonation in some of the sentences. The reasons cited for distortion were as follows:

1. The intensity and intonation did not have the natural variations required towards the end of some declarative and interrogative sentences.

2. The intensity suddenly increased or decreased in sentences at places where it was not required.

3. In some sentences, the synthesized sounds did not match their corresponding textual representations.

Table 5.5: MOS for overall perceptual quality

Naturalness			Intelligibility			Distortion		
Good	Fair	Bad	Good	Fair	Bad	Low	Medium	High
70%	30%	0%	75%	25%	0%	65%	30%	5%

This evaluation shows that the synthetic speech generated by the speech synthesizers sounds natural and is intelligible. The speech synthesizers are able to perform well even without any prosodic models. However, because some of the concatenated speech units were not appropriate, distortions appear. This brings out the need for a broader coverage of speech units in terms of units and their realizations. It is also felt that an unnatural variations in intensity and intonation need to be smoothened using appropriate techniques.

5.7 Speech synthesizers for Tamil and Hindi

Based on the evaluation and conclusions drawn from the sample speech synthesizer, Tamil and Hindi speech synthesizers were developed. The Tamil TTS has 5,740 unique units (749 monosyllables, 2,649 bisyllables and 2,342 trisyllables) with 61,274 realizations. The Hindi TTS has 8,718 unique units (1,114 monosyllables, 4,752 bisyllables and 2,852 trisyllables) with 66,788 realizations.

The primary reason for the good quality of the synthesis using syllable-like units is that they have more prosodic and acoustic information and less discontinuities when compared to other synthesis techniques using phones or diphones. As described earlier, the boundaries of the syllable-like units correspond to low energy regions of the short-term energy. These low energy regions usually correspond to minimum coarticulation points and are preferable for concatenative waveform synthesis. The speech waveform and spectrogram plot for the Tamil phrase */inda saNtAIkku mukkiya kAraNam/* is shown in Figure 5.2. The spectrogram shows that the formant changes are not abrupt at the concatenation points. Moreover, spectral changes are uniform across the syllable boundaries and hence reinforce the idea that the syllable-like unit is indeed a good candidate for concatenative speech synthesis. The number of concatenation points for such units is also very less, which is 10 for this example, instead of 26 present in the diphone synthesis. The appropriateness of the syllable-like units is evident from the naturalness of the synthesized speech. It is important to observe that models for duration, intonation or energy have not been applied. No modifications have been done on the default weights of various parameters used in calculating the target and concatenation costs in the Festival framework. This is an important difference from diphone-based synthesis that relies heavily on prosody modeling for good quality speech.

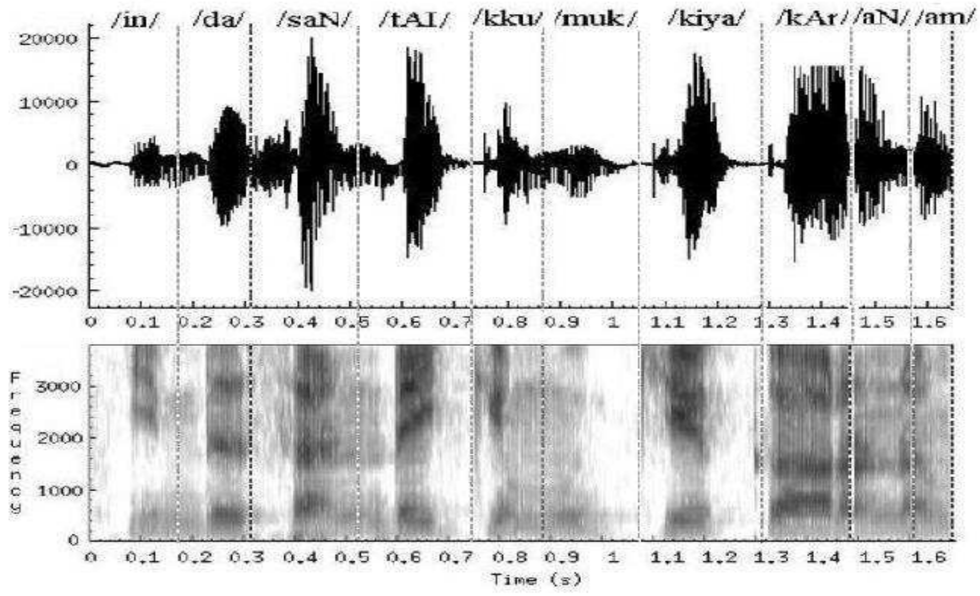

Fig. 5.2: Speech signal waveform and spectrogram for the synthesized Tamil phrase */inda saNtAIkku mukkiya kAraNam/*

Since the space requirements of concatenative speech synthesizers are fairly high - 500 MB to 700 MB, an analysis was also done on how to reduce these requirements. A set of 34,280 unique words was synthesized using the speech synthesizer to find out the most frequently used realizations of each speech unit. The infrequently used realizations were then deleted from the speech repository. This reduced the size of the speech repository of the synthesizer from 532 MB to 238 MB. The repository was further pruned by limiting the number of realizations of each speech unit to a maximum of three. This further reduced the size of the repository to 87 MB.

5.8 Issues with Tamil and Hindi speech synthesizers

Even though the speech synthesizers perform well in many cases, there are some instances where the speech quality deteriorates. An analysis was done to find reasons for this. The primary reason why the synthesized speech is poor, is the limited coverage of units for the languages for which the synthesizers have been made. Tamil and Hindi have an average 12 vowels and 22 consonants. These units can be combined to form close to 3400 monosyllable units (V, C, CV, VC, CVC) and more than two lakh polysyllables (bisyllables and trisyllables). Since it is not possible to create such a huge database of polysyllables, most words that are outside the context of the speech corpus used for creating the speech units in the database will not be synthesized using bisyllable or trisyllable units. The current speech synthesizers for Tamil and Hindi have only 749 and 1,114 monosyllables respectively. These form only about 25% of the total possible monosyllable combinations. It is hence evident that even though the syllable-like unit is a suitable candidate for unrestricted speech synthesis, good synthetic speech will not be possible until the speech databases are populated with more monosyllable units.

Another reason from poor speech quality in certain cases, is that the speaking rate and amplitude are not uniform across all the sentences from which the syllable-like units have been extracted. Some sentences have been spoken faster than other sentences. Units from these sentences produce unnatural prosodic variations in sentences in which they are used. The next section discusses how monosyllable units can be used for unrestricted speech synthesis.

5.9 Unrestricted speech synthesis using syllable-like units

Figure 5.3, shows the transcription of the speech signal corresponding to the Tamil word “vanakkam”. Three monosyllables which make the up the word - “van”,“nak” and “kam”, along with the pauses or transition periods between the syllables have been labeled. The pauses between the syllables are labeled as “*”. This example shows two aspects that need to be addressed while using monosyllable units as the basic speech units for synthesis. They are -

1. Each unit needs to have a minimum number of realizations in the speech database. This set of realizations should represent the speech characteristics of the speech unit when it appears in different positions in a word. In this example, to synthesize “vanakkam”, an instance of “van” as it appears in the beginning on a word, an instance of “nak” as it appears in the middle of a word and an instance of “kam” as it appears at the end of a word are required to produce good synthetic speech.

2. There is a finite pause between each syllable. In this example, there is a pause of 0.004 seconds between “van” and “nak” and 0.1 seconds between “nak” and “kam”. These pause durations need to be modeled so that when the syllables are concatenated together to produce synthetic speech, pauses are also present. The transitions can be represented as silences in synthetic speech.

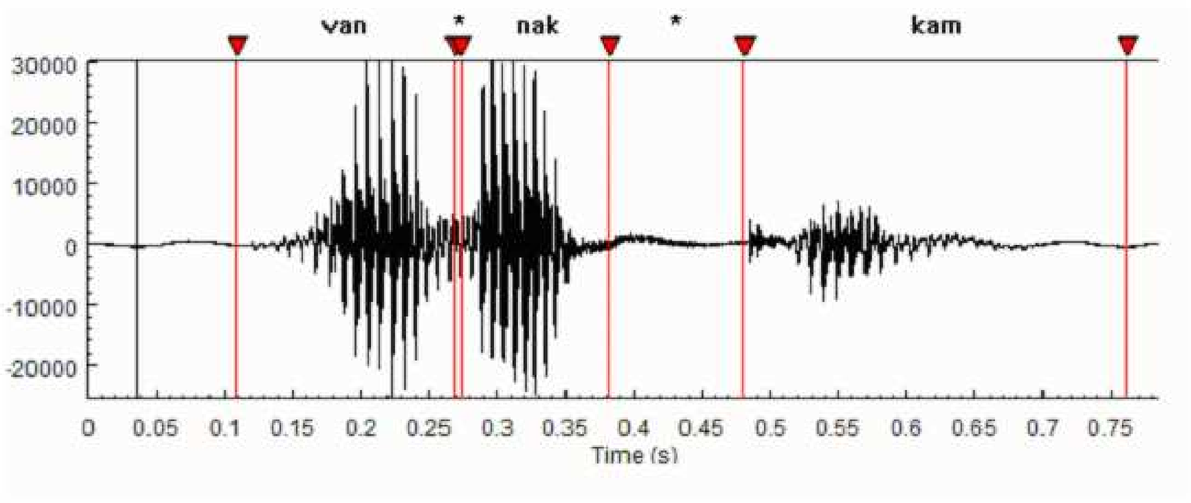

Fig. 5.3: Transcription of the speech signal corresponding to the Tamil phrase "vanakkam"

In Section 5.8.1, the minimum number of realizations required for each syllables is discussed. Section 5.8.2, explains how the pauses between the syllable units can be modeled.

5.9.1 Minimum number of realizations for syllable units

A set of 500 Tamil sentences were labeled at the monosyllable level to analyze the formant frequencies and durations of different realizations of various speech units. An example is described in this section. The formant frequencies of 10 different realizations of the monosyllable "kum" appearing in different word positions (beginning, middle and end) in the Tamil database are given in Table 5.6.

The average values of each formant at different word positions are plotted in Figure 5.4. From the plots it is observed that the formant frequencies increase as the position of the syllable moves from beginning of the word to the middle. The frequencies then start decreasing as the position shifts from the middle to the end. These characteristics

Table 5.6: Formant frequencies of different realizations of syllable "kum"

Realization	F1 (KHz)	F2 (KHz)	F3 (KHz)	F4 (KHz)	Position in word
1	319.44	942.38	2576.59	3573.93	Beginning
2	330.76	991.79	2454.41	3524.64	Beginning
3	314.60	1875.62	2807.49	3708.55	Beginning
4	322.19	1662.19	2754.034	3731.35	Beginning
5	326.71	1641.99	2344.28	4002.32	Middle
6	576.01	1803.36	2784.90	4191.44	Middle
7	641.46	1915.25	2926.18	4121.41	Middle
8	520.70	1868.48	2962.44	3862.75	End
9	444.97	1706.09	2862.78	3890.89	End
10	418.83	1594.89	2809.40	3896.20	End

are observed in most of the monosyllable units. It is hence evident that a minimum of three realizations - one realization occurring at the word beginning position, one occurring at middle of a word and another at the word ending position, are required to represent the formant variations of each speech unit.

5.9.2 Modeling pause duration between syllables

The transition time or pause duration that occurs between syllables has to be modeled to allow syllables to be joined in the same way as they occur in natural speech. To analyze the pauses that occur in natural speech between syllables, a set of 500 Tamil sentences are hand labeled. The syllables are labeled in the same manner as the group delay based segmentation algorithm identifies syllables. Units are marked as VCs or

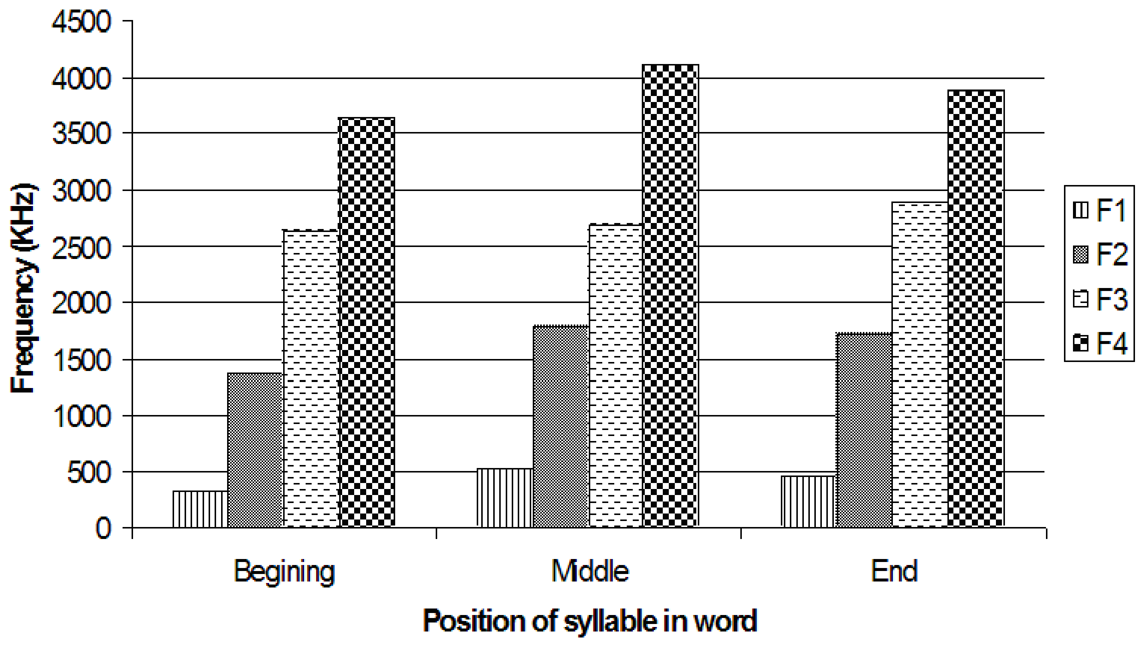

Fig. 5.4: Average formant values of syllable “kum” at different word positions

CVs at the beginning and end of words depending on how they appear. In other cases, the syllables are marked at CVC boundaries (C represents a consonant and V a vowel). The transition periods between the syllables were then marked. It is observed that the pause duration between two adjacent syllables is related to the nature of articulation of the boundary sounds of the syllables. For example, consider a CVC speech unit having constants C_1, C_2 at its boundaries. The speech unit can be represented as $C_1V_1C_2$. The adjacent speech unit can similarly be represented as $C_3V_2C_4$. It is seen that the transition period or pause duration between these adjacent syllables, $C_1V_1C_2$ and $C_3V_2C_4$ is dependent on the manner of articulation of the consonants C_2 and C_3. To analyze these characteristics, the sounds units that are found in the speech database are classified as shown in Table 5.7. The sounds are represented in their transliterated English form.

Table 5.7: Classification of basic sounds depending on manner of articulation

Basic Sounds	Class	Class Id	Manner of articulation
k kh g gh ng	Velar	1	Articulated with the back part of the tongue against the velum
c ch j jh	Palatal	2	Articulated with the body of the tongue raised against the palate
t d dh n	Retroflex	3	Articulated with tip of the tongue curled back against the palate
th	Dental	4	Articulated with the teeth
p ph b bh m	Labial	5	Articulated with the lips
y r l v	Approximant	6	Articulatory organs form a narrow channel, but leave enough space for air to flow without much audible turbulence
sh s h zh	Fricative	7	Produced by forcing air through a narrow channel made by placing two articulators close together. The articulators can be the teeth, lip or palate
a e i o u	Vowel	8	Voiced sounds with no constriction in the vocal tract

The database used for analyzing the transition periods is prepared from a set of 500 Tamil sentences. Each speech unit is assigned the class id of the class to which its boundary sound belongs to. A few entries of speech units in the database are shown in Table 5.8. The first instance in Table 5.8 shows the scenario of syllables “yil” and “dir” occurring as adjacent units. Since “l” is the boundary sound on the first syllable, “yil”, the unit is assigned to class 6 which corresponds to approximants. The adjacent syllable “dir” has “d” as its boundary sound and is assigned to class 3 which corresponds to retroflexes.

Table 5.8: Examples of database entries used for analyzing pause durations

Occurrence scenario	First syllable	Class label	Second syllable	Class label	Transition period (secs)
yil-dir	yil	6	dir	3	0.145251
pit-taar	pit	3	taar	3	0.103132
kam-kat	kam	5	kat	1	0.102679
er-kann	er	6	kann	1	0.049567

An analysis of the pause duration between syllables based these classes in explained in the following sections. In each of the sections, an analysis is done with reference to boundary sound of the second syllable of two adjacent speech units. For example, if $C_1V_1C_2$ and $C_3V_2C_4$ were adjacent speech units, the analysis is done to see how the transitions periods vary between the class corresponding to C_3 and other examples of classes 1-8 that appear adjacent to it.

5.9.3 Pause duration between velar sounds and other classes

An analysis was done on the examples of speech units where the boundary sound belongs to the velar class. Figure 5.5 shows the average, minimum and maximum values the pause duration takes for examples of the other classes. It is seen that the pause duration takes values between 0.01 seconds and 0.2 seconds for most of the classes. If the adjacent sound is a labial sound, the transition period is however only between 0.04 seconds and 0.1 seconds.

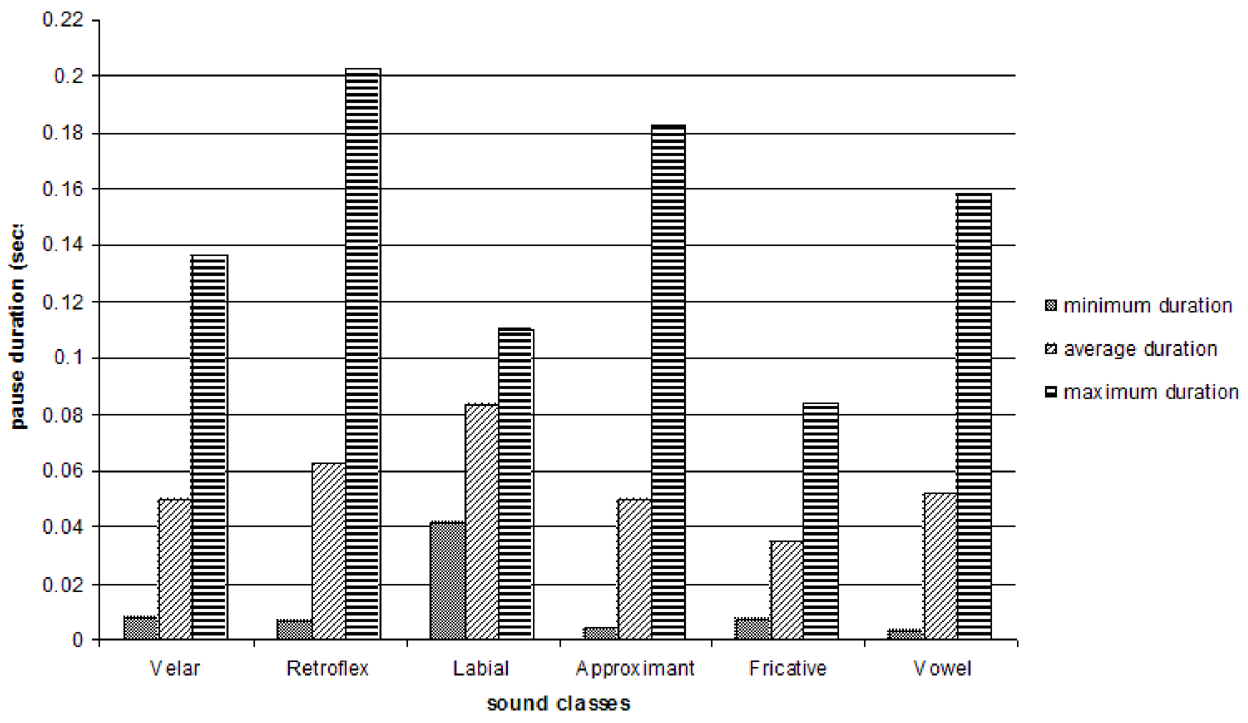

Fig. 5.5: Pause duration between velar sounds and other classes

5.9.4 Pause duration between palatal sounds and other classes

Figure 5.6 shows the average, minimum and maximum values the pause duration takes for examples between palatal sounds and the other classes. Unlike the case of velar sounds the range of values for different classes is lower and is between 0.12 seconds and 0.01 seconds. With most of the classes having examples with transition periods between 0.02 and 0.06 seconds, the average duration is around 0.03 seconds.

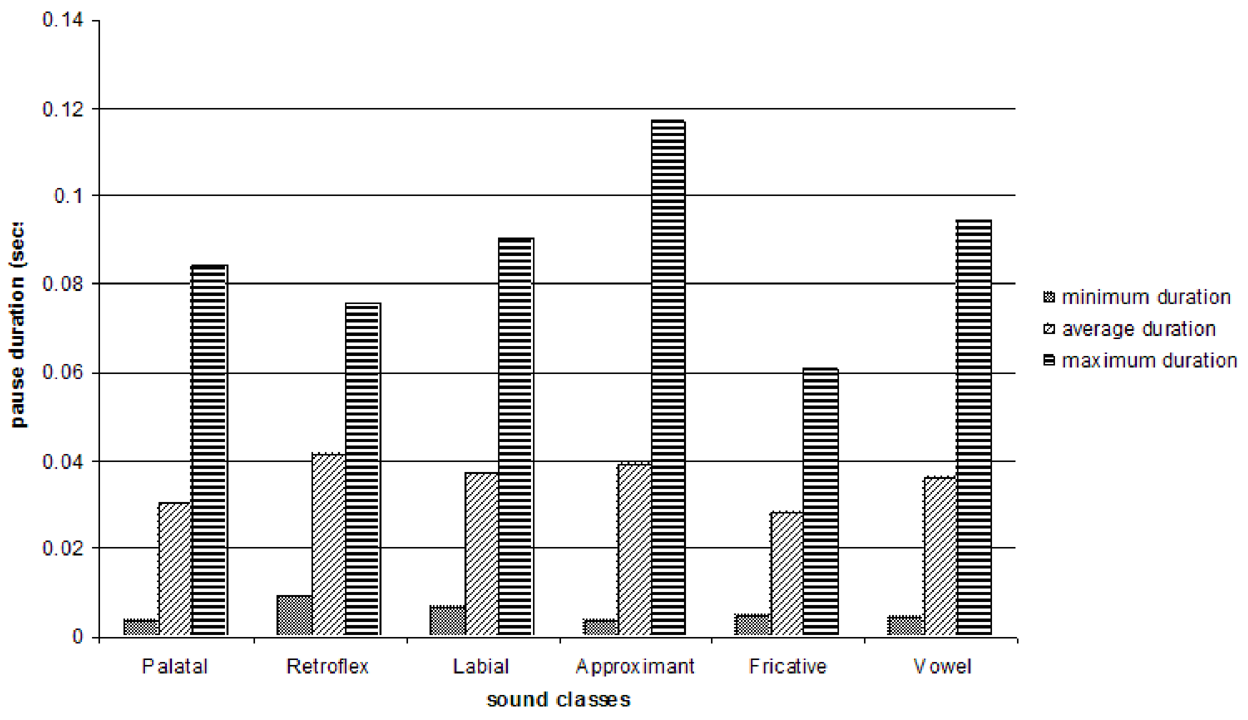

Fig. 5.6: Pause duration between palatal sounds and other classes

5.9.5 Pause duration between retroflex sounds and other classes

Figure 5.7 shows the average, minimum and maximum values the pause duration takes for examples between retroflex sounds and the other classes. The average pause duration between retroflex, labial, approximant and fricative sounds in Figures 5.6 and 5.7 are quite similar probably because of the similarity in the place of articulation. The range of the pause duration is however larger in this case, extending from 0.01 to 0.2 seconds. The pause duration appears to be more when the boundary sounds of the adjacent syllable are either vowels or approximant sounds.

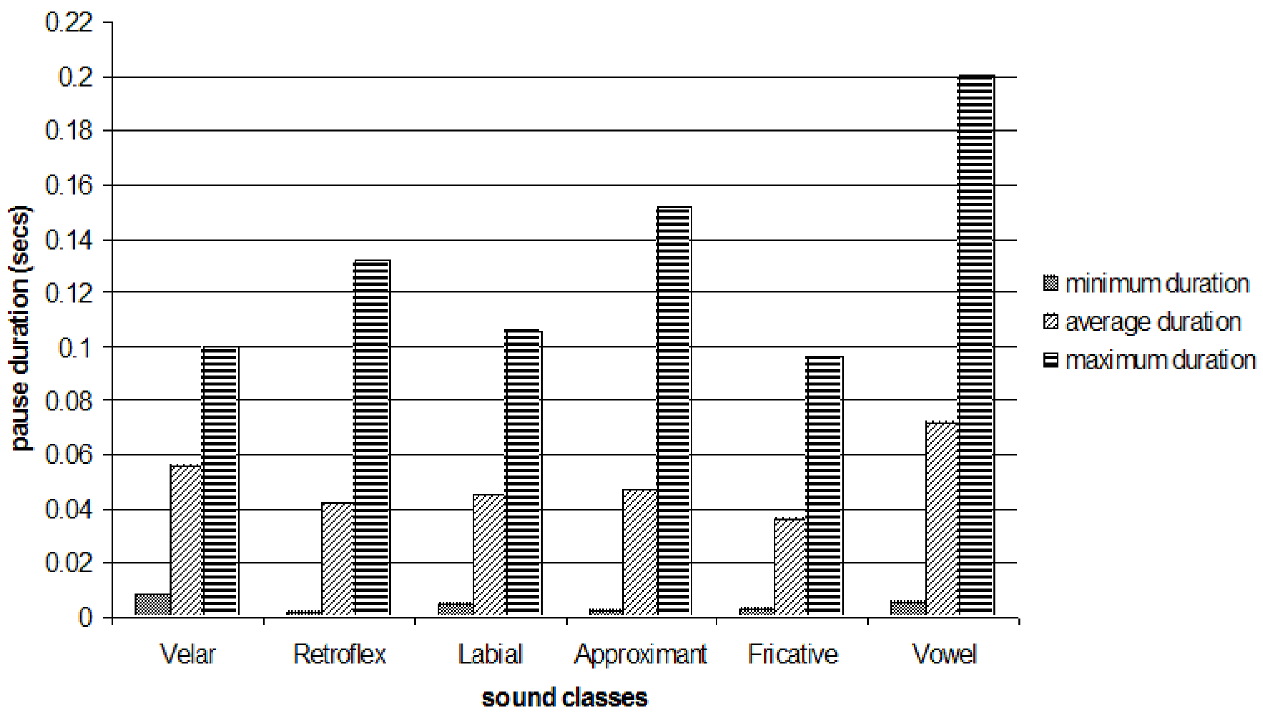

Fig. 5.7: Pause duration between retroflex sounds and other classes

5.9.6 Pause duration between dental sounds and other classes

Most of the examples of adjacent boundary sounds for dental sounds are either retroflex or labial. The other examples have been put together and average, minimum and maximum values the pause duration takes is shown in Figure 5.8. Two distinct ranges for transition periods are observed in this case. While most examples of retroflex, dental, approximant and vowel sounds have transition periods between 0.04 and 0.12 seconds, the range for labial sounds is much lower and is between 0.01 and 0.06 seconds. The average transition duration for labial sounds is hence 0.04 seconds when compared to average values of 0.06 and 0.08 seconds for the other classes.

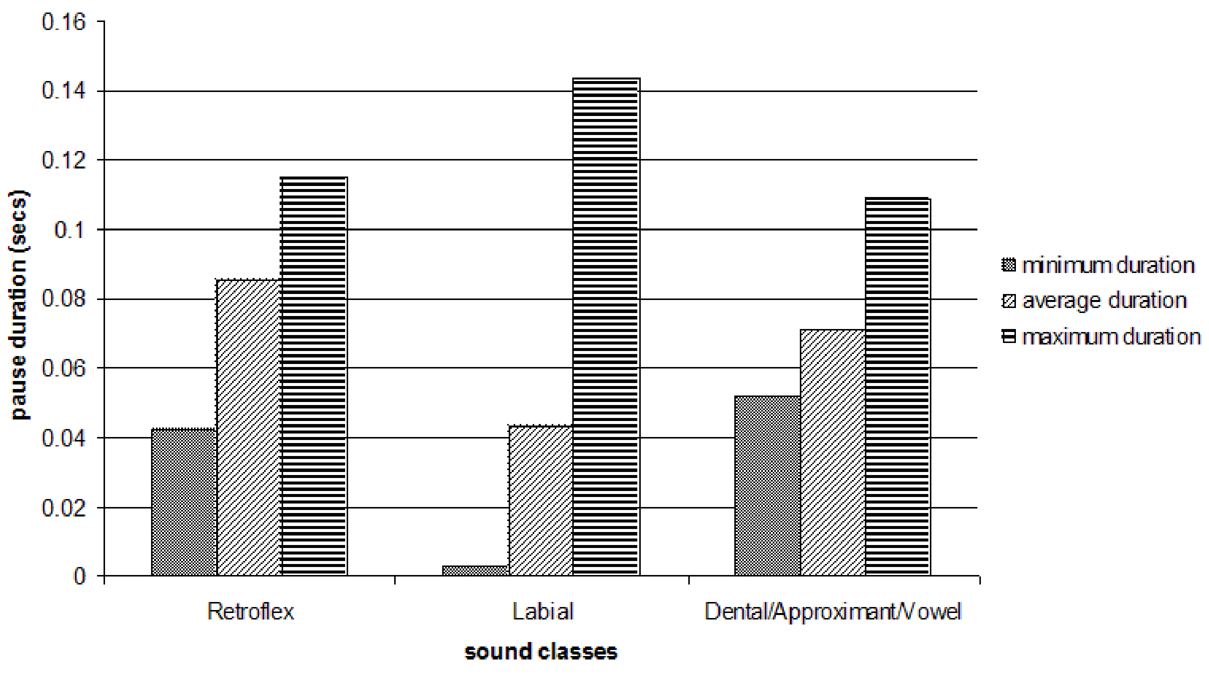

Fig. 5.8: Pause duration between dental sounds and other classes

5.9.7 Pause duration between labial sounds and other classes

Figure 5.9 shows the average, minimum and maximum values the pause duration takes for examples between labial sounds and the other classes. When compared to other classes described above, the transition period in this class is higher and has a maximum duration range between 0.1 and 0.25 seconds. All the classes show very similar trends with most of the examples having average pause durations close to 0.06 seconds.

5.9.8 Pause duration between approximant, fricative sounds and other classes

The transition periods between approximant and fricative sounds against other classes are shown in Figures 5.10 and 5.11 respectively. The average duration of these classes are similar and both of them have the lowest transition periods between adjacent sounds. In both the groups, the average pause duration is around 0.02 seconds for most of the classes when compared to other groups where the transition period is usually above 0.02 seconds.

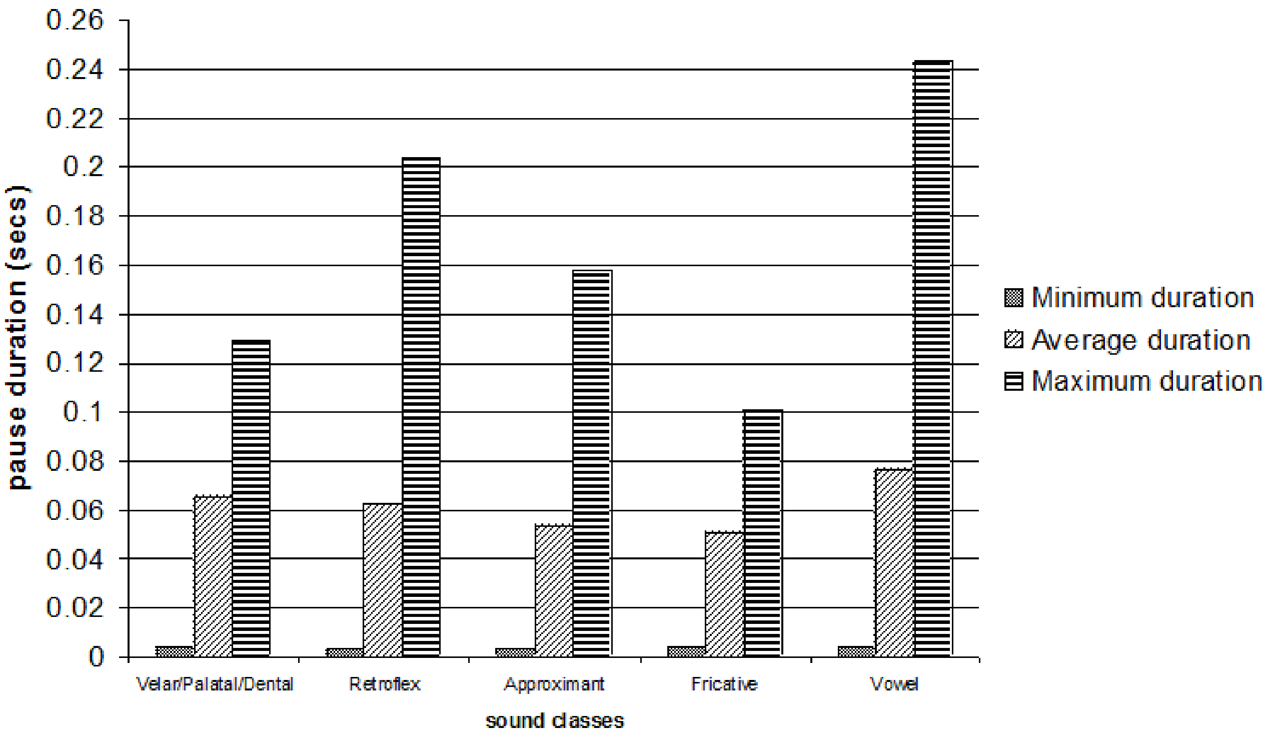

Fig. 5.9: Pause duration between labial sounds and other classes

5.9.9 Pause duration between vowel sounds and other classes

The average, minimum and maximum pause duration observed between vowel sounds against other classes is show in Figure 5.12. A number of examples have transition periods above 0.05 seconds, when compared to examples of other classes. The large transition period when compared to other classes, could be because of the time required for the vocal tract system to change from a close or semi-closed configuration while

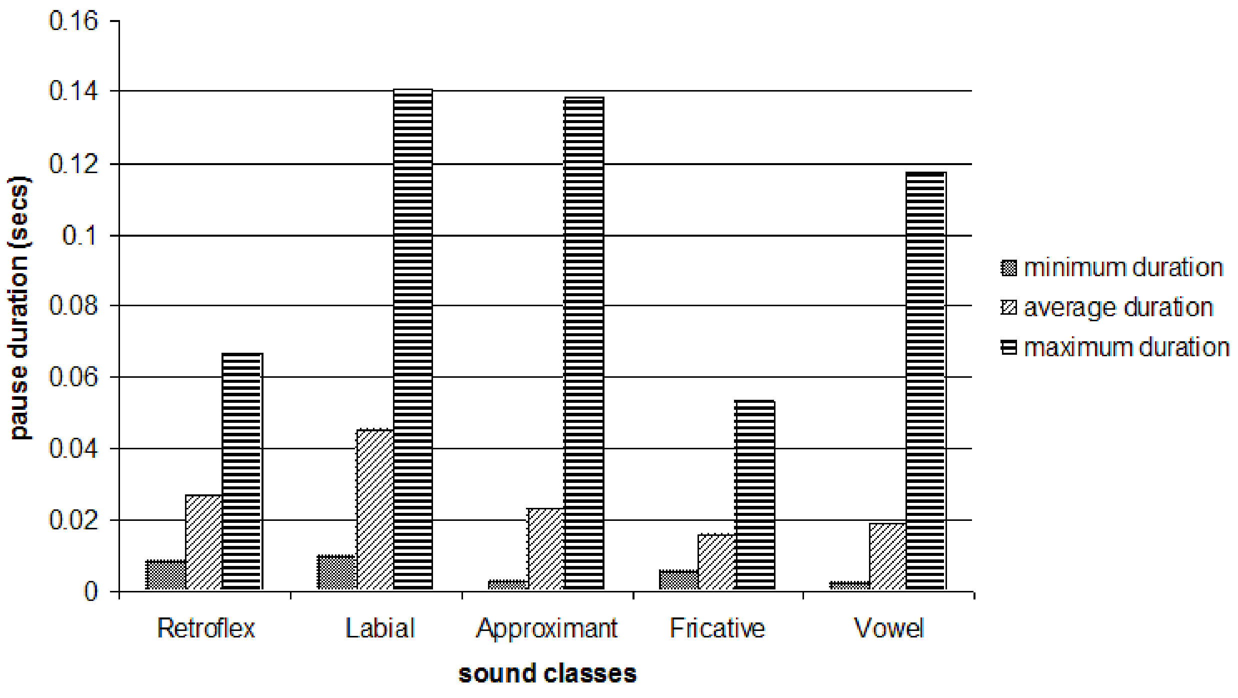

Fig. 5.10: Pause duration between approximant sounds and other classes

producing a consonant, to an open configuration to produce a voiced sound.

It should be however understood that since syllables have been split at consonant (C) boundaries, no word is split such that a vowel (V) appears in the middle of a word. Two Vs appear adjacent to each other only when the last syllable of a word is a CV and the first syllable of next word is a VC. The pause between two Vs that is being modeled is hence always a pause between two words which have vowels at their boundaries and does not represent a pause that can appear inside a word between its constituent syllables.

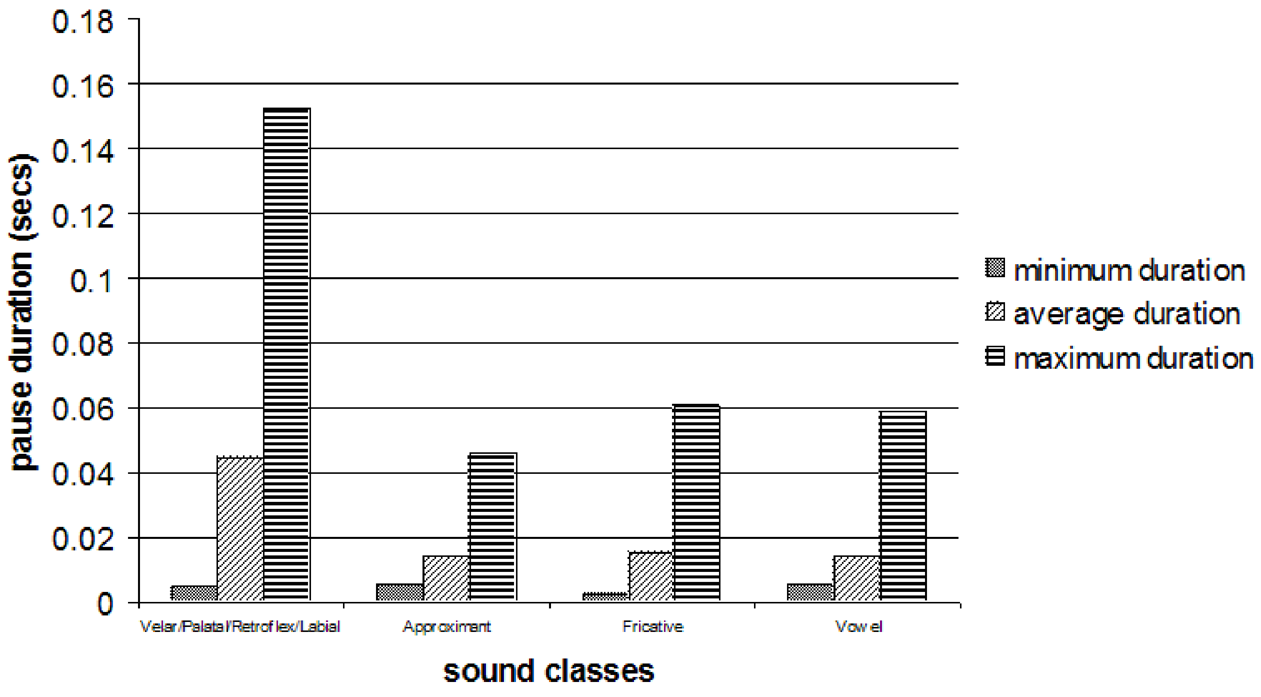

Fig. 5.11: Pause duration between fricative sounds and other classes

5.9.10 CART model for pause durations

From the discussion in the above section it is evident that pause durations differ based on which class the boundary sounds belong to. The behavior of these classes can be modeled using Classification and Regression Trees (CART). The CART model can then be used to predict the transition period between two syllables.

A CART model is built using 3959 examples of different speech unit combinations obtained from hand labeling 500 Tamil sentences. The data is prepared in the format shown in Table 5.8 The advantage of using this format is that, when the transition period between a pair of speech units needs to be predicted, either the class id information or the name of the speech units can be used. If the pair of speech units is

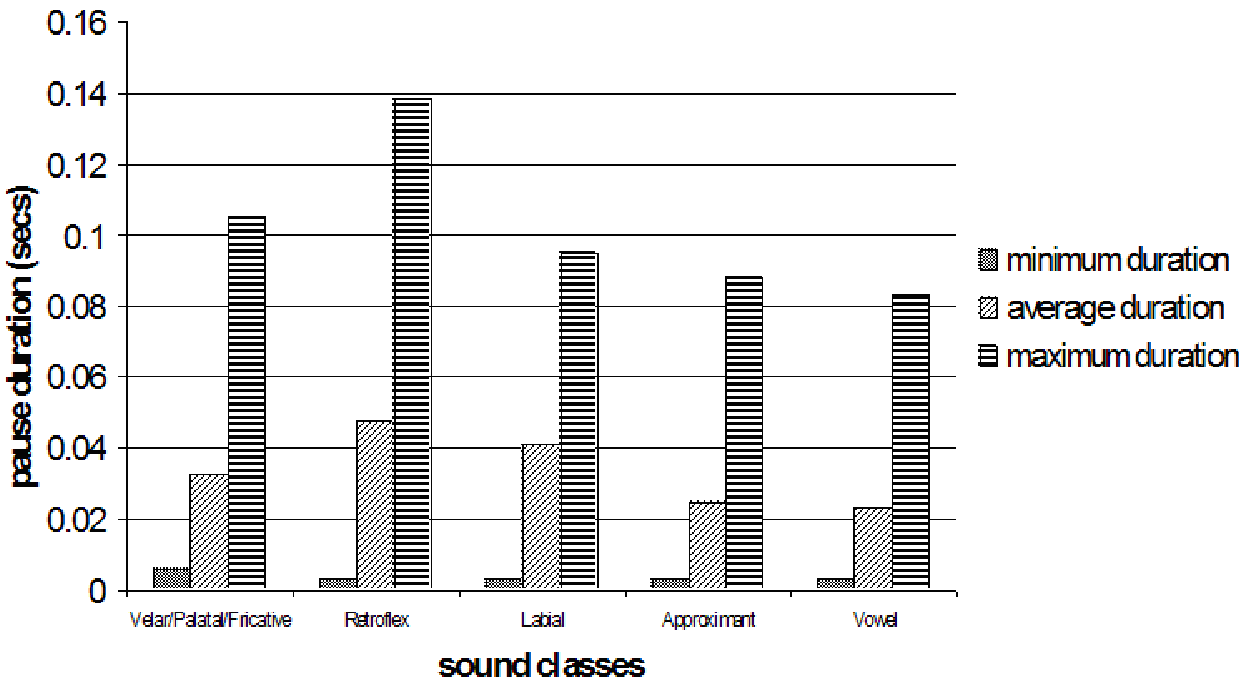

Fig. 5.12: Pause duration between vowel sounds and other classes

already seen during the training of the CART tree, the names of the speech units are used to predict the pause duration. If the pair is a new instance, the class id information is used. In this case the general behavior of the class would be outputted. The CART tree is generated using a CART building program 'wagon' [44] and is given in Appendix B. The tree has an overall correlation of 78.2%.

5.10 Conclusions

In this chapter the steps followed to build Tamil and Hindi synthesizers were discussed. An evaluation of the speech units and the synthesizers was also presented. The results of the evaluation show the effectiveness of the speech unit. Two important issues in using syllable-like units for unrestricted speech synthesis have been discussed. It has been seen that at least three realizations of speech units need to be present to represent a speech unit in the speech database. The transition periods between syllables has also been investigated and found to be dependent on the nature of articulation of boundary sound units. The transition periods are then modeled using a CART tree. In the next chapter, issues in building speech synthesizers for embedded devices are discussed.

CHAPTER 6

Distributed Speech Synthesis

6.1 Introduction

With the proliferation of mobile devices and emergence of ubiquitous computing, there is now an increasing need for speech interfaces on low-end portable devices. Many of these mobile devices do not have support for visual interfaces or data entry but have access to data through their network connectivity. Examples of these devices include PDAs, mobile phones, digital diaries, and information points on cars and home appliances. Text-to-speech synthesis (TTS) is a vital component of these speech interfaces, where low bandwidth text is converted into speech on these devices. It also finds applications in situations where a visual interface for embedded devices is inappropriate, as in the case of devices to be used by the visually or physically challenged, and the illiterate.

As described in the previous chapters, the extent of naturalness of synthetic speech produced by state-of-the-art speech synthesizers is mainly attributed to the use of concatenative synthesis. For unrestricted, close to natural speech synthesis, these synthesizers running on high end machines have RAM and hard disk requirements ranging from 500 MB to 700 MB. These requirements are mainly due to speech repositories and lexicons which hold examples of speech units with different prosodic variation. On the other hand, embedded devices have major constraints with their low memory resources and computing capabilities when compared to machines for which text-to-speech synthesizers are primarily designed. Typically an embedded device that would require a speech interface would have 100 KB to 6 MB of ROM, 100 KB to 25 MB of RAM and 5 to 50 MIPS (millions of instructions per second) of computing power. Of these resources, with only a part being available for user programs, it becomes necessary to build text-to-speech synthesizers which require low computing power and small memory footprints. The simplest choice to port these large synthesizers for embedded devices by bringing down the repository sizes is a poor option as this reduces the number of available speech units and eventually prevents unrestricted speech synthesis. This approach can however be used for limited text-to-speech applications. It is in this context that Distributed Speech Synthesis (DSS) has been suggested as an alternative, as it still allows for unrestricted speech synthesis on embedded devices with the limited memory and computational capabilities that they have.

Distributed speech synthesis is a client server approach to speech synthesis where some low memory and computational tasks are done at the embedded device or client end and the remaining memory and computational intensive tasks are done at a server. The waveform repositories and lexicons are also kept at the server end. This approach however introduces an intermediate component - a network, to transfer intermediate data between the server and the client. In many cases a network component is already present on the embedded devices and the DSS merely needs to connect and use the mobile network or computer network to which these devices are already connected to. Unlike its counterpart in speech recognition called Distributed Speech Recognition (DSR) which has been studied and standardized by the ETSI STQ Aurora DSR Working Group [50], DSS has not been formalized. The DSS involves various technologies apart from text-to-speech synthesis - distributed computing and system design, network transmissions and protocols, multi-threaded programming, XML and Web techniques which necessitate a detailed study on DSS [25].

In this chapter, the DSS systems are built for two speech synthesis systems - the Festival system, a large speech synthesis system already described in detail in this thesis, and Flite [51] a small, fast, run-time synthesis library suitable for embedded applications. Even though Flite is designed as an alternative run-time synthesis platform for Festival, it has a fairly large footprint requiring about 6-10 megabytes of RAM depending on the language. The focus of this chapter is to illustrate methods by which both these types of synthesizers can be run from embedded devices by using a client-server approach to distribute tasks. The discussions in the following sections use mobile phones as an example. It should be noted that these methods are not only suitable for embedded devices but also for stand alone systems with low memory and computing capabilities.

6.2 The Design of the DSS system

The design of DSS system primarily involves the analysis of the synthesis cycle used by the speech synthesizer planned to be split into a client and server, and identifying which stages of the cycle can be present at the client and server ends. With the DSS introducing a slow network between the client and server, it also becomes necessary to find how to speed up the process by using techniques like multi-threading and buffering.

6.2.1 Analysis of the Festival Synthesis Cycle

Festival provides a highly flexible and modular architecture for TTS conversion. A basic installation of the framework using diphone synthesis takes about 30 MB space on disk and 25 MB of memory when executed on a Pentium machine. Using other synthesis techniques like cluster unit selection for high quality speech synthesis with Festival requires more than 500 MB of disk space to store speech units [51]. From the memory and computing capabilities of certain popular Nokia mobile phones, given in Table 6.1 [52], it is evident that it will not be possible to run the framework on most commonly used mobile phones as they have only limited memory and computing resources. However these mobile phones are enabled to operate on data networks like GSM-CSD (Global System for Mobile Communications - Circuit Switched Data), GPRS (General Packet Radio Service), HSCSD (High Speed Circuit Switched Data) or EDGE (Enhanced Data rates for GSM Evolution) which have speeds of 9.6-14.4 kbps(kilobits per second), 32-48 kbps, 57.6 kbps and 384 kbps respectively [53].

An appropriate design of a DSS system for mobile phones taking all the above mentioned - limited memory, low computing power and a slow network connection into consideration, would require a client-server approach designed appropriately to suit the computational and memory requirements of the client machine and the bandwidth of the connecting network. The design of a DSS introduces a division of tasks of the typical speech synthesis cycle of speech synthesizers as shown in Figure 1.2, between a DSS client and server. This cycle needs to be split into two parts - a part with low memory and computing requirements that will form the DSS client, and a part that requires large computing and memory requirements to form the DSS server. This is illustrated in Figure 6.1 as three phases.

The first phase accepts text from users for synthesis, performs basic steps of pro-

Table 6.1: Popular Nokia mobile phones, their memory, processor and data network support details.

Nokia Model	Memory	Processor	Data Network Support
3230	6 Mb	120 Mhz	GPRS/EDGE
7650	3.6 MB	104 Mhz	GPRS/HSCSD
6600	6 Mb	104 MHz	GPRS/HSCSD
6620	12 Mb	150 MHz	GPRS
6630	10 Mb	220 Mhz	WCDMA/GPRS/EDGE
7710	90 Mb	150 Mhz	GPRS/EDGE/HSCSD

cessing, and sends an intermediate form to the DSS server over the network for further processing. The middle phase of synthesis on the server processes the intermediate form, carries out the remaining steps of synthesis and sends back speech data to the client. After receiving back the intermediate speech data generated by the server, the final phase constructs a synthetic waveform as the output of synthesis. To ensure that the memory requirements of the embedded device are met, these stages need to be designed such that the DSS client does not require any speech databases. Keeping the DSS client language independent also allows end users to use speech synthesis in various languages using a single client running on their mobile by only switching between servers that provide the services. The stages must also be designed such that there is minimum data transfer between the server and client over the slow network, which could otherwise become a bottleneck if the size of intermediate data transfers is large.

Festival's speech synthesis cycle involves various stages similar to those outlined

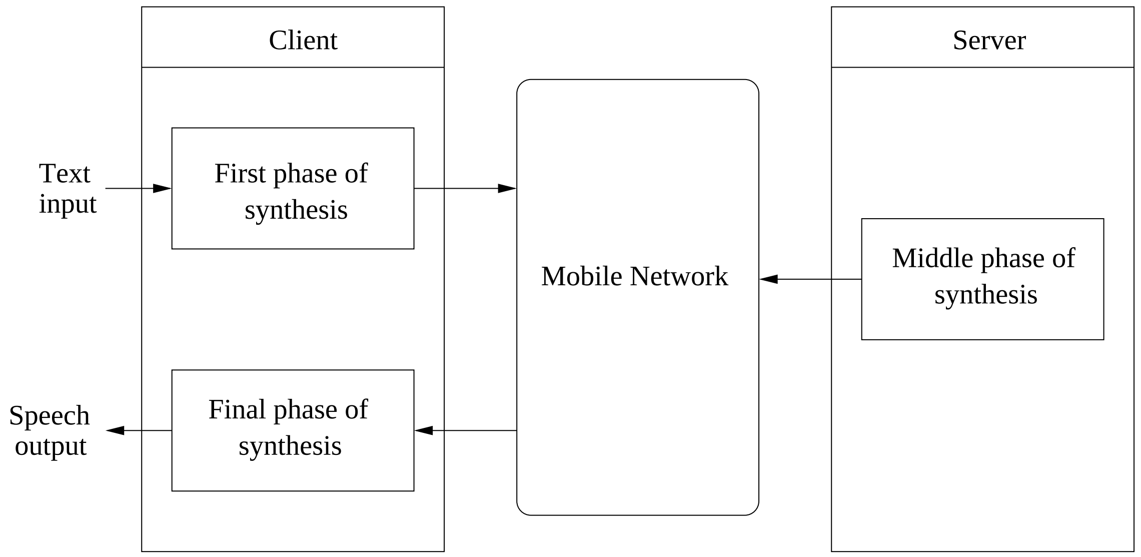

Fig. 6.1: Schematic diagram of a DSS system

in Figure 1.2 with each stage being implemented in C++ or Scheme [24] . Festival uses a data structure called an *'utterance'* along with these stages or *modules*, as they are called in the Festival terminology. An *'utterance'* structure starts with text input from the user and ends with a wavefile after passing through each of the modules. The various modules of Festival are listed below for convenience of explanation:

1) *Token_POS* module does basic token identification and is used for homograph disambiguation,

2) *Token* module applies token-to-word rules,

3) *POS* module applies part-of-speech rules,

4) *Phrasify* module does prosodic phrasing using statistically trained models and CART trees,

5) *Word* module performs a lexical lookup to find pronunciations from a lexicon or letter-to-sound rules,

6) *Pauses* module predicts pauses based on rules,

7) *Intonation* module predicts accents and boundaries using rules,

8) *PostLex* modules applies post lexicon rules if any,

9) *Duration* module predicts duration of segments using statistically trained models or CART trees,

10) *Int_Targets* module generates F_0 values for each identified speech unit,

11) *Wave_Synth* modules generates the synthesized waveform for the input text.

Of these, modules till the *POS* stage do not require any large databases or involve complex operations. In addition to its speech synthesis capabilities, Festival provides a number of utility functions too. Two such functions which are of use in our context are as follows: the function that saves *'utterance'* structures as text files and the function that allows an *'utterance'* structure to be recreated from text files. Festival further allows each module to be invoked individually as functions on an *'utterance'* structure instead of using the default synthesis methods which take an input text through the entire speech synthesis cycle. Using these features, the DSS system can be visualized as follows - a DSS client that performs the (*Token_POS*, *Token*) stages of synthesis and writes an *'utterance'* structure to disk. This *'utterance'* structure is then transfered over the network to the DSS server which loads the *'utterance'* structure into the framework. Instead of using the default speech synthesis functions, the server now applies the remaining speech synthesis functions on the *'utterance'* structure one after another till completion. The synthesized waveform is then compressed and sent back to the client for play back at the client. This DSS implementation ensures that the memory requirement of the DSS client is within 100 KB and can be executed with the computational capabilities of the mobile phones.

6.2.2 Streaming Synthesis for the Festival system

A straight forward implementation of the client and server as described in the previous section, is to have the client accept text from the user and pass it through the first two stages of the speech synthesis cycle. The *'utterance'* structure after the first two stages is then sent to the server over the mobile network. The server loads the semi-processed *'utterance'* structure and continues with the remaining stages of processing. The final waveform is then compressed and sent back to the client. After the entire wave file is received at the client, the playback to the user begins. However, this simple implementation is bound to result in a considerably large delay between the time the user enters text at the client running on the embedded device and the time the speech output is available because of data moving twice across the intermediate slow network introduced by the DSS. It is therefore essential to design the client and server for the DSS with suitable techniques to reduce this undesired delay. Two suitable designs are discussed in this section. In both these designs the input text is no longer processed as a single entity but is first split into sentences or short phrases. Each of these sentences or phrases is then processed individually. Both of the designs essentially try to pipeline the following seven independent tasks in the DSS process:

1) creating *'utterance'* structures from sentences or phrases of the input text after the first two stages of the synthesis structure,

2) sending the *'utterance'* structures from the client to the server,

3) receiving *'utterance'* structures at the server,

4) performing the remaining stages of synthesis at the server and compressing the waveforms at the server,

5) sending the waveforms back to the client,

6) receiving the waveform at the client,

7) playing the waveforms back to the user.

The first design uses a single thread to execute each task. For example, the design has an *'utterance'* creation thread for creating *'utterance'* structures for each sentence by passing the sentence through the first two phases of synthesis, a network sending thread to send the *'utterances'* to the server and so on. The threads keep passing on tasks as they finish. For example, the *'utterance'* creation thread passes on the created *'utterance'* structure for a sentence to the network sending thread at the client as soon as it finishes with it, and then picks up the next sentence for processing. Reusable buffers are used to hold intermediate data passed from a preceding thread if a particular thread is still busy with a previous task. By using a data queue between the playback thread and the network receiving thread at the client, speech data gets buffered even when a received waveform is being played. There is an initial latency corresponding to the synthesis time for the first sentence. The advantage of this method is that the user starts receiving speech data after a short delay when compared to the straight forward implementation because individual sentences or phrases are being handled instead of the entire input text. Delays in between phrases or sentences also appear because playback for a waveform is over before the next waveform has been fully received. The intermediate delay may not be acceptable if there is a mix of long and short sentences or a large number of long sentences in the input text. Randomly dividing the input text to make equally sized sentences to reduce intermediate delays has adverse results on the intelligibility of the output because the input text does not get broken at its natural sentence boundaries.

The second design tries to reduce the delay introduced by the straight forward implementation by using multiple threads which perform the synthesis of each sentence in parallel. In this design, for each of the sentences or phrases identified, an assigned

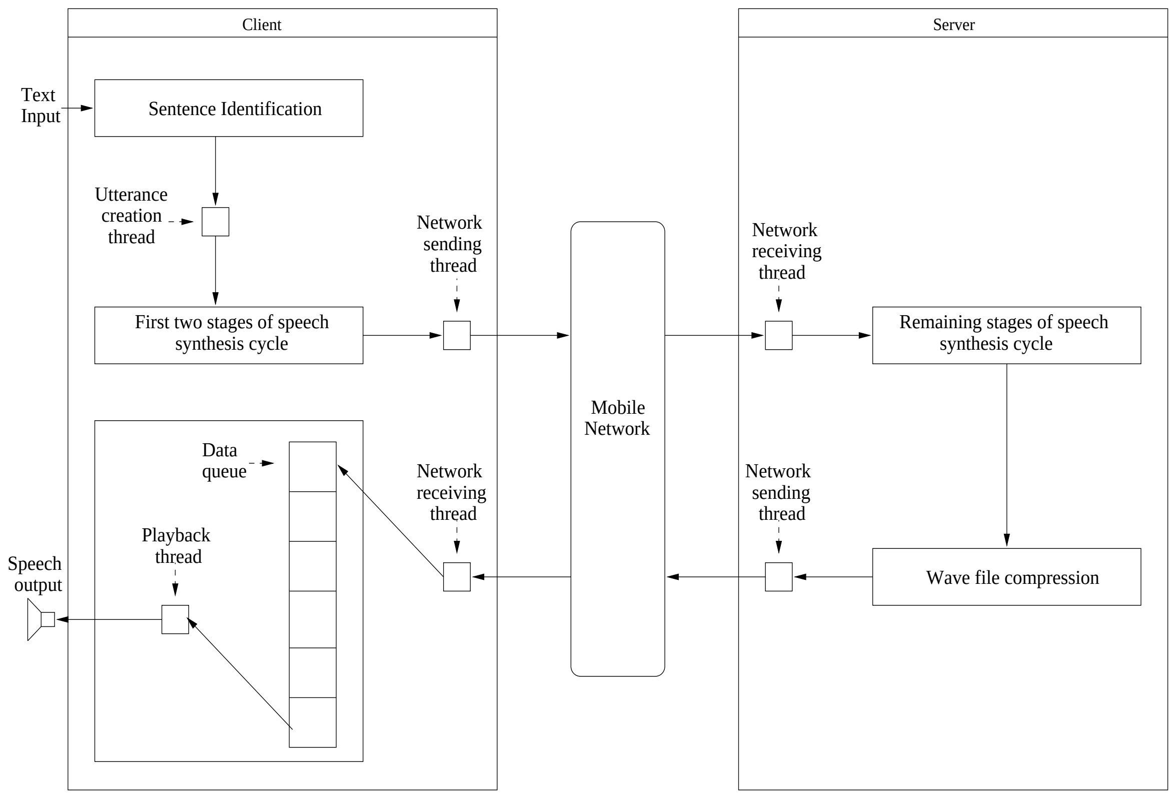

Fig. 6.2: Single thread DSS implementation

thread handles all the tasks involved with the sentence - processing the sentence, transmitting its *'utterance'* structure to the server and receiving back the waveform corresponding to the sentence at the client. Each client thread has a counterpart at the server end that handles the remaining stages of synthesis, compression of the waveform and network transmission. Sentences complete synthesis out of order based on sentence length and hence need to be buffered till synthesis of all sentences is complete. While this design reduces the latency present in the straight forward implementation, it still has an initial latency equal to the synthesis time of the largest sentence or phrase of the input text but no intermediate delays. However if the server is not powerful enough to synthesize all the sentences of a text in parallel, this will increase the initial latency.

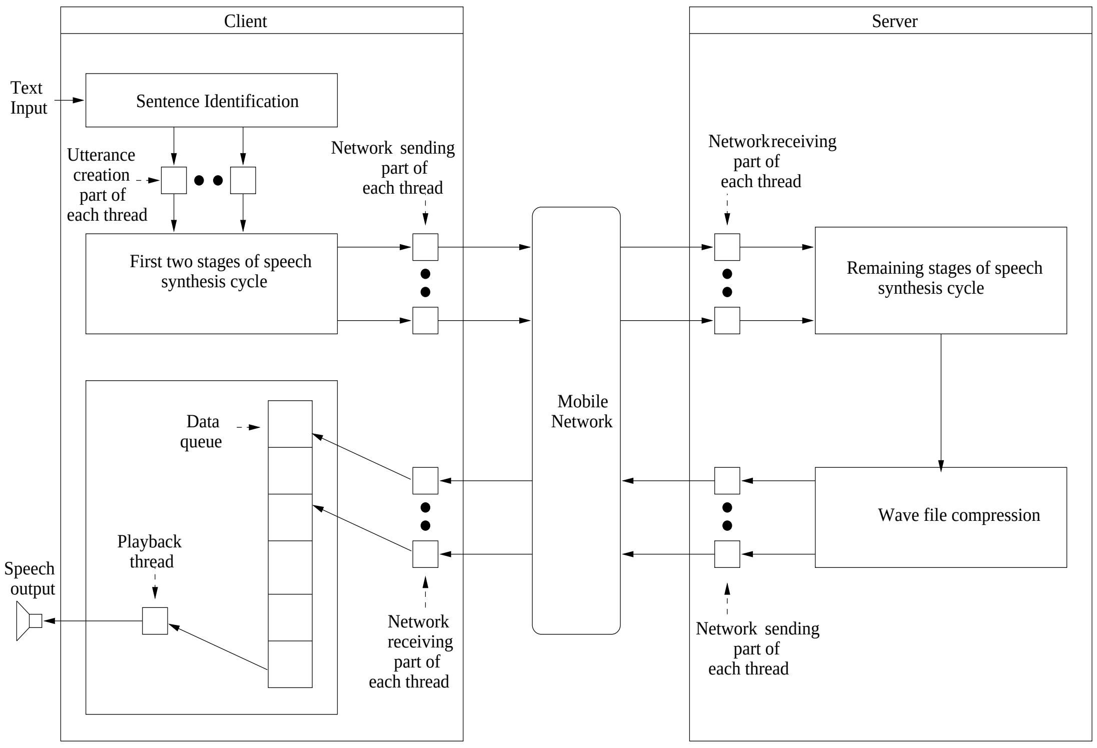
Client
Server
Text Input
Sentence Identification
Utterance creation part of each thread
Network sending part of each thread
First two stages of speech synthesis cycle
Data queue
Playback thread
Speech output
Network receiving part of each thread
Mobile Network
Networkreceiving part of each thread
Remaining stages of speech synthesis cycle
Wave file compression
Network sending part of each thread

Fig. 6.3: Multiple thread DSS implementation

6.2.3 Analysis of the Flite Synthesis Cycle

The Flite system consists of the following four modules:

1. a core library module: C objects that are used to build the Flite system,

2. a language module: language definitions in the form of sound units of the language (phones), tokenization rules,

3. a voice module: models for speaker-specific prosody and intonation,

4. a language lexicon module: letter-to-sound rules and pronunciation models for out-of-vocabulary words.

The Flite system for American English with a size of 6 MB is used in this work. The various stages of speech synthesis in the Flite system are given in Table 6.2. The three phases as shown in Figure 6.1 are to be formed by combining one or more stages to make a phase.

Table 6.2: Contribution of different stages to the total code size and network traffic using Flite

Phase	Stage	Contribution to total interstage data transfer for synthesis (%)	Contribution to total code size(%) and prominent contributor to size
I	Tokenization of text	7.3	1.1 Text Parsers
	Token analysis	2.7	
	Part of speech tagging	1.0	
	Building phrasing relationships	0.03	
II	Creating relationships with words	13.7	35.2 Language Lexicons
	Prediction of pauses,		
	Applying intonation		
III	Application of post lexicon rules Duration prediction of sentences	5.7	63.6 Diphone databases
	Creating the F0 contour	10.9	
	Generation of the waveform	59.7	
IV	Transfer of PCM values and playback	Variable − depends on the text to be synthesized	0.1 Network components

Table 6.2 shows that any design of client and server applications is basically a trade off between the code size permitted using available memory resources and the network bandwidth available. The client would essentially be required to take in the text, get it synthesized and do a playback. It would accept text from users for synthesis, perform text tokenization on the input text and send an intermediate form to the server over network for further processing. The server processes the tokens, carries out

the remaining intermediate steps in synthesis and sends back the speech data to the client. After receiving back the speech data generated by the server, the client finally constructs the waveform to be played out as the output of synthesis. By offloading the steps that take up most of the memory requirements from the client as illustrated in the first architecture given above, the code size reduces to about 64KB, a size that is acceptable on many embedded systems. The server accounts for about 5.5 MB.

6.3 Conclusions

The designs for the DSS system are presently implemented and tested on desktop machines. Apart from the DSS server, the client and mobile network are only simulations. The DSS server is implemented in C++ and uses the Festival API set for C/C++ to interface to the Festival synthesis framework. The DSS client is built with Symbian C++ using the Series 60 2.1 Platform Software Development Kit(SDK) from Nokia for the Symbian OS 7.0s. The Platform SDK has a rich set of APIs for implementing the client designs discussed in the previous section. The implementation runs using the emulator available with the Platform SDK [26]. The single threaded distributed speech synthesizer was tested on a Wireless Local Loop (WLL) based IP network with a low bandwidth of 70Kbps to find the minimum network requirements for effective network operations.

A performance evaluation based on Mean Opinion Scores (MOS) on a 5-point scale with 20 listeners using two paragraphs of text gave an acceptability score of 75%. The two paragraphs of text had a total number of 24 sentences with 10 to 70 characters per sentence. The time taken for text-to-speech synthesis by a standalone system and the distributed synthesis system is given in Table 6.3. It is seen that the time taken by the distributed synthesis system is about five times more than that of the standalone system. The mean opinion scores are low primarily because of the prominent gaps that appear in synthesis of long sentences. The acceptability rate may be increased

Table 6.3: A comparison of time (in seconds) taken by DSS over a low bandwidth network and a standalone system

Number of characters in sentence	Standalone system	Distributed synthesis system using a 70 Kbps WLL link
10	0.65	5.22
20	1.40	6.89
30	2.09	9.48
40	2.57	11.92
50	2.98	13.82
60	3.21	16.87
70	4.19	21.03

by dynamically dividing large sentences at suitable points. Increasing the network bandwidth can also reduce the time required for synthesis. Both the systems are now implemented on high-end machines and need to be implemented on embedded systems for which they are meant. Issues related to actual implementation needs to be addressed. These results could also lead to the development of an ASIC for text-to-speech systems for embedded systems.

CHAPTER 7

Summary and Conclusion

7.1 Summary of the work

The primary objective of this work was to identify a speech unit for Indian languages that is better than the commonly used subword units like diphones, and use it along with a suitable speech synthesis technique. Such a speech unit and its use with an alternative speech synthesis technique are required because the existing speech synthesis techniques for Indian languages require extensive prosodic modeling. Owing to unavailability of adequately large and properly annotated databases for most Indian languages, prosodic models for these synthesizers have still not been developed properly. With poor prosodic models in place, the quality of synthetic speech generated by these synthesizers is poor. Unit selection based synthesis provides an alternative to this by using portions of recorded speech and concatenating them together. Speech units picked by the selection algorithm optimally minimize both a target cost (how close a database unit is to the desired unit) and a join cost (how well two adjacently selected units join together) and hence when concatenated together produce speech with adequate prosody. Speech synthesizers using this technique also do not need extensive prosodic modeling. In this thesis the objective has hence been to find a suitable speech unit that could be used along with the unit selection synthesis technique to produce natural sounding synthetic speech for Indian languages.

With initial experiments showing that syllables are a good choice for speech synthesis for Indian languages, the focus has been on identification of syllable units. In this thesis, a group delay based segmentation algorithm has been used to identify "syllable-like" units from continuous speech data. The group delay based segmentation algorithm was useful in identifying speech units with less manual effort, with a good consistency and with a performance close to that of manual segmentation. Integrating these speech units with the Festival speech synthesis framework and using the unit selection algorithm implemented in the framework led to the development of speech synthesizers for Tamil and Hindi. These speech synthesizers are capable of generating natural sounding synthesized speech with no prosodic modeling. It has been shown in this thesis that using "syllable-like" units up to bisyllables strikes a balance between the quality of speech synthesized and the number of units needed for creating a speech database. It has also been demonstrated that this approach is general and is suitable for other languages as well. Development of these TTS systems also involved the creation of a comprehensive set of letter-to-sound rules and an elaborate phoneset for both Hindi and Tamil. An analysis on how these speech units can be used for unrestricted speech synthesis has also been done.

This thesis has addressed the issues in building speech synthesizers for embedded devices. Distributed speech synthesis has been suggested as a means by which unrestricted speech synthesis can be made available on such devices. The architectural design and implementation mechanism for DSS systems based on two synthesizers - Festival and Flite, have been presented in the thesis.

7.2 Criticism of the work

The price for the naturalness in synthetic speech using unit selection concatenation techniques is the lack of control over the prosody of the generated speech [54]. In real applications, the database may not have units that match both desired spectral and prosodic features.

It is evident that having large amounts of data in a unit selection database would lead to better synthesis because it now becomes more likely that a unit closer to the target and more likely to have a better join would be present [55]. It is important not just to collect every possible unit but instead to collect the “right” set of units which are most frequently used in a particular language along with their prosodic variations. This requires a “phonetically balanced” set of sentences to be used for creating the speech database. Designing such a database allows speech units to have a uniform number of realizations in various prosodic variations instead of having a skewed database with certain units having a large number of realizations while some units have only a few realizations. The Tamil and Hindi synthesizers developed in this thesis have not been created using a phonetically balanced set of sentences. The speech corpus should also be recorded such that the energy levels, rate of speech and pitch remain uniform across all the units. Speech units should also be properly labelled taking into account co-articulation effects with neighbouring units.

7.3 Future Directions

Some form of F_0 modeling has to be done to smoothen out the unnatural changes in pitch for unrestricted text-to-speech synthesis. Time domain based algorithms to do the same need to be developed and integrated. Simple concatenation of units manually has shown to produce synthesized speech that is sometimes better that than that produced by the Festival framework. It might hence be useful to see a simpler synthesizer based on the unit selection technique can be built to replace the Festival framework, to work with these units.

The "syllable-like" unit has been shown to be a good speech unit for Indian languages. It has not been used for other languages in the Indian scenario which have high demands for TTS systems like Indian English. Its applications in such areas could be explored.

APPENDIX A

A comprehensive set of letter-to-sound rules are used in the Hindi and Tamil text-to-speech synthesizers to syllabify the input text into the syllable-like units. The letter-to-sound rules are written in Scheme, the scripting language used by the Festival framework. Festival applies the letter-to-sound rules on each word of the input text to find the speech units to be used to synthesize each of the words. Simple rules of the form `([ x y ] = xy)` are used to build the letter-to-sound rules for the languages. The rule `([ x y ] = xy)` says that if characters `‘x’`, `‘y’` appear together in a word, the speech unit represented by label `‘xy’` should be used during synthesis.

The rules written for the Hindi and Tamil TTS are applied in such a way that each word is split into its largest constituent syllable units: trisyllables first, bisyllables only if trisyllable combinations are not present, and finally monosyllables if bisyllables combinations are also not present. The Tamil TTS has 5,740 unique speech units (749 monosyllables, 2,649 bisyllables and 2,342 trisyllables). The Hindi TTS has 8,718 unique speech units (1,114 monosyllables, 4,752 bisyllables and 2,852 trisyllables). Each speech unit is assigned a unique label that corresponds to its textual representation. These labels are used to form simple rules, as illustrated above. The right hand side of each rule contains the label corresponding to a speech unit. The left hand side of each rule corresponds to the characters in the input text for which the speech unit should be used. The Hindi TTS has 8,718 rules corresponding to each of the syllable-like units identified for the language. The Tamil TTS has 5,740 rules.

The rules corresponding to each type of speech unit - monosyllables, bisyllables and trisyllables, are grouped together under separate rule sets. While monosyllables have one vowel or a combination of a vowel and a consonant, bisyllables have a combination of two vowels and two consonants. Trisyllables, are still larger units and have a combination of more than two vowels and consonants. While synthesizing text, Festival first checks if the text can be synthesized using the trisyllable rule set. If not possible, it then uses the bisyllable ruleset and finally the monosyllable rule set if no combinations can be found also in the bisyllable ruleset. A few of the rules in each of the rule sets used in the Hindi and Tamil synthesizers are given in this section.

A.1 Letter-to-sound rules for Hindi

A.1.1 Monosyllable rules for Hindi

Monosyllable rules are used to include the speech units that are obtained for smaller values of the WSF factor (4-6) (Chapter 5, Section 2). Monosyllables have one vowel or a combination of a vowel and a consonant. For Hindi, 1,114 such units have been identified. The rule `([ a b ] = ab)` says that if characters `'a'`, `'b'` appear together in a word, the speech unit represented by label `'ab'` should be used while synthesizing the characters. A sample from the 1,114 monosyllable rules is given below.

```
(lts.ruleset iit_hindi_mono
        ([ a b ] = ab)
        ([ a b h ] = abh)
        ...
        ([ b a ] = ba)
        ([ b a b ] = bab)
        ...
        ([ c h i s ] = chis)
        ([ c h i t ] = chit)
        ([ c h i v ] = chiv)
        ...
        ([ j a ] = ja)
        ([ j a b ] = jab)
```

```
            ([ j a c h ] = jach)
            ...
            ([ k i k ] = kik)
            ([ k i l ] = kil)
            ...
            ([ p a ] = pa)
            ([ p a b ] = pab)
            ([ p a c h ] = pach)
            ...
            ([ r a l ] = ral)
            ([ r a m ] = ram)
            ...
            ([ s h i l ] = shil)
            ([ s h i n ] = shin)
            ...
            ([ z r a t ] = zrat)
            ([ z u ] = zu)
            ([ z y a ] = zya)
)
```

A.1.2 Bisyllable rules for Hindi

Bisyllable rules represent the speech units that are obtained for WSF factor values between 8-12 (Chapter 5, Section 2). Bisyllables typically have a combination of two vowels and two consonants. For Hindi, 4,752 such units have been identified. The rule `([ b a h a] = baha)` says that if characters '`b`', '`a`', '`h`', '`a`' appear together in a word, the speech unit represented by label '`baha`' should be used while synthesizing the characters. A sample from the 4,752 bisyllable rules is given below.

```
    (lts.ruleset iit_hindi_bi
            ([ a b a d ] = abad)
            ([ a b a s ] = abas)
            ...
            ([ b a h a ] = baha)
            ([ b a h a d ] = bahad)
            ([ b a h a k ] = bahak)
            ...
            ([ c h a l a ] = chala)
            ([ c h a l a k ] = chalak)
            ([ c h a l a n ] = chalan)
            ...
            ([ g u r k h a ] = gurkha)
            ([ g u t h o n ] = guthon)
            ([ g v a n i ] = gvani)
            ...
            ([ j a u t ] = jaut)
            ([ j a v a ] = java)
            ([ j a v a b ] = javab)
            ...
            ([ k a i n ] = kain)
            ([ k a i p ] = kaip)
            ([ k a i s ] = kais)
            ...
            ([ m a t r i ] = matri)
            ([ m a t r u b ] = matrub)
            ([ m a u ] = mau)
            ...
            ([ p a i d ] = paid)
            ([ p a i g ] = paig)
            ([ p a i m ] = paim)
            ...
            ([ r a n a ] = rana)
            ([ r a n e ] = rane)
            ...
            ([ s a m u d ] = samud)
            ([ s a m u h ] = samuh)
            ([ s a m v a ] = samva)
            ...
            ([ z o r d a r ] = zordar)
            ([ z o r i ] = zori)
            ([ z y a d a ] = zyada)
)
```

A.1.3 Trisyllable rules for Hindi

Trisyllable rules represent the speech units that are obtained for WSF factor values between 15-18 (Chapter 5, Section 2). Trisyllables typically have a combination of more than two vowels and consonants. For Hindi, 2,852 such units have been identified. The rule `([ a b h i n e ] = abhine)` says that if characters `‘a’`,`‘b’`,`‘h’`,`‘i’`,`‘n’`,`‘e’`, appear together in a word, the speech unit represented by label `‘abhine’` should be used while synthesizing the characters. A sample from the 2,852 trisyllable rules is given below.

```
(lts.ruleset iit_hindi_tri
        ([ a b h a r a t ] = abharat)
        ([ a b h i n e ] = abhine)
        ...
        ([ b a r s a y e ] = barsaye)
        ([ b a s v a n a ] = basvana)
        ([ b a t a l i ] = batali)
        ...
        ([ c h a u t h e ] = chauthe)
        ([ c h a u t i s ] = chautis)
        ([ c h a u v a n ] = chauvan)
        ...
        ([ e l a k e ] = elake)
        ([ e l i v i j ] = elivij)
        ([ e m i l a d ] = emilad)
        ...
        ([ j a n a r d h a n ] = janardhan)
        ([ j a n a s a n ] = janasan)
        ([ j a n a t a ] = janata)
        ...
        ([ k e h a n a ] = kehana)
        ([ k e l i y e ] = keliye)
        ([ k e n d r i y a ] = kendriya)
        ...
        ([ m a h o t s a v ] = mahotsav)
        ([ m a i d a n ] = maidan)
        ([ m a i g a v a t ] = maigavat)
        ...
        ([ p u c h a n a ] = puchana)
        ([ p u n a r v a s ] = punarvas)
        ([ p u n y a t i t ] = punyatit)
        ...
        ([ r a j i y a b ] = rajiyab)
        ([ r a j n a i t ] = rajnait)
        ...
        ([ t i s a r e ] = tisare)
        ([ t i t o d i ] = titodi)
        ([ t i y o g i ] = tiyogi)
        ...
        ([ y o j a n a ] = yojana)
        ([ y o n o t i l ] = yonotil)
        ([ z i m e d a r ] = zimedar)
)
```

A.2 Letter-to-sound rules for Tamil

A.2.1 Monosyllable rules for Tamil

Monosyllable rules are used to include the speech units that are obtained for smaller values of the WSF factor (4-6) (Chapter 5, Section 2). Monosyllables have one vowel or a combination of a vowel and a consonant. For Tamil, 749 such units have been identified. The rule `([ a d ] = ad)` says that if characters `'a'`, `'d'` appear together in a word, the speech unit represented by label `'ad'` should be used while synthesizing the characters. A sample from the 749 monosyllable rules is given beloew.

```
(lts.ruleset iit_tamil_mono
        ([ a d ] = ad)
        ([ a h ] = ah)
        ([ a k ] = ak)
        ...
        ([ c h a n ] = chan)
        ([ c h a p ] = chap)
        ([ c h a r ] = char)
        ...
        ([ d a n ] = dan)
        ([ d a p ] = dap)
        ([ d a r ] = dar)
        ...
        ([ j e n ] = jen)
        ([ j e p ] = jep)
        ([ j e r ] = jer)
        ...
        ([ k a l ] = kal)
        ([ k a m ] = kam)
        ([ k a n ] = kan)
        ...
        ([ l u l ] = lul)
        ([ l u m ] = lum)
        ([ l u p ] = lup)
        ...
        ([ m u n ] = mun)
        ([ m u p ] = mup)
        ([ m u r ] = mur)
        ...
        ([ r a d ] = rad)
        ([ r a j ] = raj)
        ([ r a k ] = rak)
        ...
        ([ s h a ] = sha)
        ([ s h a l ] = shal)
        ([ s h a m ] = sham)
        ...
        ([ z u n ] = zun)
        ([ z u p ] = zup)
        ([ z u r ] = zur)
)
```

A.2.2 Bisyllable rules for Hindi

Bisyllable rules represent the speech units that are obtained for WSF factor values between 8-12 (Chapter 5, Section 2). Bisyllables typically have a combination of two vowels and two consonants. For Tamil, 2,649 such units have been identified. The rule `([ c h a i ] = chai)` says that if characters `‘c’`, `‘h’`, `‘a’`, `‘i’` appear together in a word, the speech unit represented by label `‘chai’` should be used while synthesizing the characters. A sample from the 2,649 bisyllable rules is given below.

```
    (lts.ruleset iit_tamil_bi
        ([ a c h a ] = acha)
        ([ a c h i ] = achi)
        ...
        ([ c h a d u r ] = chadur)
        ([ c h a d y a ] = chadya)
        ([ c h a i ] = chai)
        ...
        ([ d a k a r ] = dakar)
        ([ d a k i ] = daki)
        ([ d a k i l ] = dakil)
        ...
        ([ i n a r ] = inar)
        ([ i n d a ] = inda)
        ([ i n d i ] = indi)
        ...
        ([ j a m u ] = jamu)
        ([ j a n a ] = jana)
        ([ j a n d a ] = janda)
        ...
        ([ k a l a n ] = kalan)
        ([ k a l a p ] = kalap)
        ([ k a l a r ] = kalar)
        ...
        ([ m a d u ] = madu)
        ([ m a h a ] = maha)
        ([ m a i ] = mai)
        ...
        ([ p a d i r ] = padir)
        ([ p a d i s ] = padis)
        ([ p a d r i ] = padri)
        ...
        ([ r i d u l ] = ridul)
        ([ r i k a ] = rika)
        ...
        ([ s h i l a ] = shila)
        ([ s h i y a s ] = shiyas)
        ([ s i l a n ] = silan)
        ...
        ([ z u d i ] = zudi)
        ([ z u d u ] = zudu)
        ([ z u i n g ] = zuing)
)
```

A.2.3 Trisyllable rules for Tamil

Trisyllable rules represent the speech units that are obtained for WSF factor values between 15-18 (Chapter 5, Section 2). Trisyllables typically have a combination of more than two vowels and consonants. For Tamil, 2,342 such units have been identified. The rule `([ a d a l a l ] = adalal)` says that if characters `‘a’`,`‘d’`,`‘a’`,`‘l’`,`‘a’`,`‘l’`, appear together in a word, the speech unit represented by label `‘adalal’` should be used while synthesizing the characters. A sample from the 2,342 trisyllable rules is given below.

```
    (lts.ruleset iit_tamil_tri
        ([ a d a l a l ] = adalal)
        ([ a d a r a ] = adara)
        ...
        ([ c h a l a d a i ] = chaladai)
        ([ c h a l a i ] = chalai)
        ([ c h a m a c h a m ] = chamacham)
        ...
        ([ d a r k a n a ] = darkana)
```

```
          ([ d a r k a v i r ] = darkavir)
          ([ d a r p o d a ] = darpoda)
          ...
          ([ e v a r a s ] = evaras)
          ([ e z a i ] = ezai)
          ([ e z a m a ] = ezama)
          ...
          ([ j e n i v a ] = jeniva)
          ([ j e r m a n i ] = jermani)
          ([ j e y a l a ] = jeyala)
          ...
          ([ k u t a r i k ] = kutarik)
          ([ k u t a v e n ] = kutaven)
          ([ k u t i m a ] = kutima)
          ...
          ([ m a d i k a ] = madika)
          ([ m a d i p i l ] = madipil)
          ([ m a d i p u ] = madipu)
          ...
          ([ p a d a r k u ] = padarku)
          ([ p a d a t a d ] = padatad)
          ([ p a d a v i ] = padavi)
          ...
          ([ r i p a i ] = ripai)
          ([ r i r u k i ] = riruki)
          ...
          ([ t i r u p a ] = tirupa)
          ([ t i y a k a ] = tiyaka)
          ([ t o k u p u ] = tokupu)
          ...
          ([ y i r a d i ] = yiradi)
          ([ y i r u k i ] = yiruki)
          ([ z u v i n a r ] = zuvinar)
)
```

APPENDIX B

A CART tree generated using '`wagon`' (CART tree building tool), is used to predict the duration of pauses between syllables when they are concatenated together. The tree uses 4 features to predict the duration of pauses between the two adjacent syllables - the class id of the syllable `(syl_class)`, the name of the syllable `(syl_name)`, the class id of the adjacent syllable `(p_syl_class)` and the name of the adjacent syllable `(p_syl_name)`. The tree is accessed during run time from the Festival framework with corresponding values for each of these features using a function called '`wagon_predict`'. Each invocation uses a feature vector of the form - `((syl_name` *`name`*`) (syl_type` *`type id`*`) (p_syl_name` *`name`*`) (p_syl_type` *`type id`*`))`. An example feature vector is `((syl_name vit) (syl_type 6) (p_syl_name dith) (p_syl_type 4))`. The function returns a duration value based on these features. A sample of nodes from the 1,923 node CART tree is given below.

```
;;Cart Tree
            (p_syl_class is 6)
            (p_syl_class is 5)
            (syl_name is kum)
            (0.000519016 0.016684)
            (syl_name is vam)
            (0.00772485 0.0243053)
            (syl_name is lam)
            (1.34343e-05 0.0240095)
            (syl_name is lum)
            (0.034231 0.058867)
            (0.0474964 0.0560409)
            (syl_name is nnur)
            (0.00737866 0.0903145)
            (p_syl_class is 3)
            (syl_name is yenn)
            (0.0108604 0.0393183)
            (syl_name is mann)
            (0.00378316 0.033589)
            (syl_name is chenn)
```

```
(0.00561655 0.0157755)
(p_syl_name is rue)
(0.00253547 0.031979)
(syl_name is inn)
(0.00580597 0.0194958)
(syl_name is vonn)
(0.00733906 0.0311265)
(syl_name is yinn)
(0.00570345 0.0212703)
(syl_name is munn)
(0.00806899 0.023259)
(syl_name is monn)
(0.00949998 0.0246325)
(syl_name is menn)
(0.0100764 0.0234647)
(syl_name is rann)
(0.0145402 0.0194515)
(syl_name is daann)
(0.0121686 0.0248535)
(p_syl_name is ru)
(0.0148156 0.0217463)
(p_syl_name is riy)
(0.0323084 0.0352415)
(0.0166058 0.0250118)
(syl_name is pper)
(0.00596021 0.0750795)
(p_syl_name is vir)
(p_syl_class is 6)
(0.0123558 0.0168474)
(0.0274136 0.0681827)
(syl_name is var)
(0.0378999 0.0288588)
(syl_name is maar)
(0.0415282 0.0383752)
(p_syl_name is vit)
(0.0685122 0.0384098)
(syl_name is rue)
(0.0692342 0.068492)
(p_syl_name is vil)
(0.0210428 0.0232188)
(p_syl_class is 8)
(p_syl_name is yinn)
(0.0316697 0.0230996)
(p_syl_name is vaz)
(0.00965551 0.017983)
(syl_name is zu)
(0.00651882 0.0184235)
(syl_name is diy)
(0.00140646 0.00655837)
(syl_name is kiy)
(0.00173226 0.00567722)
(syl_name is lai)
(0.00196266 0.0046534)
(p_syl_name is var)
(0.0679629 0.060549)
(p_syl_name is yak)
(0.00199501 0.00592178)
(syl_name is triy)
(0.00190276 0.00557117)
(p_syl_name is yann)
```

```
(0.00186874 0.007063)
(p_syl_name is yam)
(0.00167163 0.00663838)
(p_syl_name is yay)
(0.000298399 0.004444)
(p_syl_name is yar)
(0.00238779 0.00723385)
(p_syl_name is yad)
(0.00221195 0.00673063)
(p_syl_name is yai)
(0.00145238 0.00695343)
(p_syl_name is ya)
(0.00230983 0.0068385)
(syl_name is liy)
(0.00184163 0.00640825)
(syl_name is kaiy)
(0.00148872 0.006082)
(p_syl_name is yat)
(0.00146858 0.00625033)
(syl_name is chiy)
(0.00148592 0.0071548)
(syl_name is nniy)
(0.00200957 0.00642267)
(p_syl_name is yem)
(0.00148422 0.0056855)
(p_syl_name is yenn)
...
(syl_name is kal)
(0.0478782 0.0385062)
(0.0222705 0.023692)
(syl_name is aar)
(0.00334815 0.0465755)
(syl_name is bil)
(0.017486 0.0375975)
(syl_name is tum)
(0.0208495 0.0462368)
(syl_name is dar)
(0.029865 0.0501866)
(p_syl_name is div)
(0.0301171 0.036797)
(p_syl_class is 1)
(syl_name is kal)
(0.0128481 0.023785)
(0.0299087 0.0486515)
(syl_name is tar)
(0.00345846 0.0117935)
(p_syl_class is 5)
(p_syl_name is nniy)
(0.0453786 0.0388685)
(0.0332672 0.0470986)
(syl_name is kil)
(0.0379677 0.038774)
(syl_name is per)
(0.0438311 0.051353)
(p_syl_name is nnat)
(0.092474 0.073255)
```

REFERENCES

[1] S. Furui, *Digital Speech Processing, Synthesis and Recognition.* Marcel Dekker, 2001.

[2] T. Dutoit, *An Introduction to Text-to-Speech Synthesis.* Kluwer Academic Publishers, 1997.

[3] M.D. Riley, "Tree-based modeling for speech synthesis," *Talking Machines: Theories, Models and Designs*, pp. 265–273, 1992.

[4] D. Jurafsky and J.H. Martin, *Speech and Language Processing.* Pearson Education, 2000.

[5] P. Rubin, T. Baer and P. Mermelstein, "An articulatory synthesizer for perceptual research," *Journal of the Acoustical Society of America*, vol. 70, pp. 321–328, 1981.

[6] D. Klatt, "Software for a cascade/parallel formant synthesizer," *Journal of the Acoustical Society of America*, vol. 67, pp. 971–995, 1980.

[7] A.W. Black and K.A. Lenzo, "Building synthetic voices." `http://festvox.org/bsv/`, 2003.

[8] F. Charpentier and M. Stella, "Diphone synthesis using an overlap-add technique for speech waveforms concatenation," in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 2015–2018, 1986.

[9] J. Laroche, Y. Stylianou and E. Moulines, "HNS: Speech modification based on a harmonic+noise model," in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 550–553, 1993.

[10] A.J. Hunt and A.W. Black, "Unit selection in a concatenative speech synthesis system using a large speech database," in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 1, pp. 373–376, 1996.

[11] A.W. Black and P. Taylor, "Automatically clustering similar units for unit selection in speech synthesis," in *Proceedings of EUROSPEECH*, pp. 601–604, 1997.

[12] S.P. Kishore, R. Sangal and M. Srinivas, "Building Hindi and Telugu voices using Festvox," in *Proceedings of International Conference on Natural Language Processing*, pp. 18–21, 2002.

[13] S.R. Rajeshkumar, *Significance of Duration Knowledge for a Text-to-Speech System in an Indian Language.* M.S. Dissertation, Department of Computer Science and Engineering, Indian Institute of Technology Madras, 1990.

[14] B. Yegnanarayana, S. Rajendran, V.R. Ramachandran and A.S. Madhukumar, "Significance of knowledge sources for a text-to-speech system for Indian languages," *Sadhana*, pp. 147–169, 1994.

[15] A. Sen and K. Samudravijaya, “Indian accent text-to-speech system for web browsing,” *Sadhana*, pp. 113–126, 2002.

[16] G.L. Jayavardhana Rama, A.G. Ramakrishnan, R. Muralishankar and P. Prathibha, “A complete text-to-speech synthesis system in Tamil,” in *Proceedings of IEEE Workshop on Speech Synthesis*, pp. 191–194, 2002.

[17] S.P. Kishore, R. Kumar and R. Sangal, “A data driven synthesis approach for Indian languages using syllable as basic unit,” in *Proceedings of International Conference on Natural Language Processing*, pp. 311–316, 2002.

[18] N.Sridhar Krishna and H.A. Murthy, “Duration modeling of Indian languages Hindi and Telugu,” in *Proceedings of ISCA Speech Synthesis Workshop*, pp. 197–202, 2004.

[19] Z. Hu, J. Schalkwky, E. Baranrd and R. Cole, “Speech recognition using syllable-like units,” in *Proceedings of Int. Conf. Spoken Language Processing*, vol. 2, pp. 1117–1120, 1996.

[20] S. Greenberg, “Understanding speech understanding: Towards a unified theory of speech perception,” in *Proceedings of ESCA Workshop on the Auditory Basis of Speech Perception*, pp. 1–8, 1996.

[21] A. Hasuenstein, “Using syllables in a hybrid HMM-ANN recognition system,” in *Proceedings of EUROSPEECH*, vol. 3, pp. 1203–1206, 1997.

[22] S.P. Kishore and A.W. Black, “Unit size in unit selection speech synthesis,” in *Proceedings of EUROSPEECH*, pp. 1317–1320, 2003.

[23] V.K. Prasad, *Segmentation and Recognition of Continuous Speech*. Ph.d. dissertation, Department of Computer Science and Engineering, Indian Institute of Technology Madras, 2002.

[24] A.W. Black, P. Taylor and R. Caley, “The Festival speech synthesis system.” `http://festvox.org/festival/`, 1998.

[25] H. Tang, B. Yin and R. Wang, “Study on distributed speech synthesis systems,” in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 732–735, 2003.

[26] R. Harrison, *Symbian OS C++ for Mobile Phones*. John Wiley, 2003.

[27] ITU-T Recommendation P.85, “A method for subjective performance assessment of the quality of speech voice output devices,” 1994.

[28] E. Keller, F. Bailly and A. Monaghan, *Improvements in Speech Synthesis, COST 258: The Naturalness of Synthetic Speech*. John Wiley, 2002.

[29] J. Bachenko and E. Fitzpatrick, “Computational grammar of discourse-neutral prosodic phrasing in English,” *Computational Linguistics*, vol. 16, pp. 155–170, 1990.

[30] R. Willemse and L. Boves, “Context free wild card parsing in a text-to-speech system,” in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 3757–760, 1991.

[31] M. S. J. Allen and D. Klatt, *From Text-to-Speech: The MITalk System.* Cambridge University Press, 1987.

[32] S. Lee and Y.H. Oh, "Tree-based modeling of prosodic phrasing and segmental duration for Korean TTS systems," *Speech Communication*, vol. 28, pp. 283–300, 1999.

[33] N. Sridhar Krishna, *Text-to-speech synthesis system for Indian languages within the Festival Framework.* M.S. Dissertation, Department of Computer Science and Engineering, Indian Institute of Technology Madras, 2004.

[34] H. Fujisaki and S. Ohno, "Analysis and modeling of fundamental frequency contour of English utterances," in *Proceedings of EUROSPEECH*, pp. 985–988, 1993.

[35] K. Silverman, M. B. Beckman, J. Pirelli, M. Ostendorf, C. Wightman, P. Price, J. Pierrehumbert and J. Hirschberg, "Tobi: A standard for labeling English prosody," in *Proceedings of Int. Conf. Spoken Language Processing*, pp. 867–870, 1992.

[36] D. O'Shaughnessy, "A multispeaker analysis of durations in read French paragraphs," *Journal of the Acoustical Society of America*, vol. 76, pp. 1664–1672, 1984.

[37] K. Bartkova and C. Sorin, "A model of segmental duration for speech synthesis in French," *Speech Communication*, vol. 6, pp. 245–260, 1987.

[38] N. Kaiki, K. Takeda, Y. Sagisaka, "Statistical analysis for segmental duration rules in Japanese text-to-speech," in *Proceedings of Int. Conf. Spoken Language Processing*, pp. 17–20, 1990.

[39] W.N. Campbell and D. Isard, "Segment durations in a syllable frame," *Journal of Phonetics*, vol. 19, pp. 37–47, 1991.

[40] J.C. Lee, J. Kang, D. Kim and S. Sung, "Energy contour generation for a sentence using a neural network learning method," in *Proceedings of Int. Conf. Spoken Language Processing*, pp. 1991–1994, 1998.

[41] P.C. Bagshaw, "Unsupervised training of phone duration and energy models for text-to-speech synthesis," in *Proceedings of Int. Conf. Spoken Language Processing*, pp. 17–20, 1998.

[42] S. Lemmetty, *Review of Speech Synthesis Technology.* M.S Dissertation, Laboratory of Acoustics and Audio Signal Processing, Helsinki University of Technology, 1999.

[43] M. Hunt, D. Zweirynski and R. Carr, "Issues in high quality LPC analysis and synthesis," in *Proceedings of EUROSPEECH*, pp. 348–351, 1989.

[44] P. Taylor, R. Caley and A.W. Black, "The Edinburgh speech tools library, 1.2.1 edition." `http://www.cstr.ed.ac.uk/projects/speech_tools`, 2002.

[45] M. Macon, A. Cronk, J. Wouters and A. Kein, "OGIresLPC: Diphone synthesizer using residual-excited linear prediction." Tech. Rep. CSE-97-007, Department of Computer Science, Oregon Graduate Institute of Science and Technology, 1997.

[46] H.A. Murthy, "The real root cepstrum and its applications to speech processing," in *Proceedings of National Conference on Communication*, pp. 180–183, 1997.

[47] H.A. Murthy and B. Yegnanarayana, "Formant extraction from minimum phase group delay functions," *Speech Communication*, vol. 1, pp. 209–221, 1991.

[48] T. Nagarajan and H. Murthy, "Subband-based group delay based segmentation of spontaneous speech into syllable-like units," *EURASIP Journal of Applied Signal Processing*, vol. 17, pp. 2614–2625, 2004.

[49] *Database for Indian languages.* IIT Madras, Chennai, India: Speech and Vision Lab, 2001.

[50] W. Zhang, L. He, Y. Chow, R. Yang and Y. Su, "The study on distributed speech recognition systems," in *Proceedings of IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 1431–1434, 2000.

[51] A.W. Hunt and K. Lenzo, "Flite: a small fast run-time synthesis engine," in *Proceedings of 4th ISCA speech synthesis workshop*, pp. 157–162, 2001.

[52] Nokia, "Device Specifications." `http://www.forum.nokia.com/devices`, 2005.

[53] T.S. Rappaport, *Wireless Communications: Principles and Practice.* Prentice Hall, 2001.

[54] A. Raux and A.W. Black, "A unit selection approach to F0 modeling and its application to emphasis," in *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*, pp. 700–705, 2003.

[55] A.W. Black, "Perfect synthesis for all of the people all of the time," in *Proceedings of of the IEEE Workshop on Speech Synthesis*, pp. 157–162, 2002.

LIST OF PUBLICATIONS

1 Samuel Thomas, Hema A. Murthy and C. Chandra Sekhar, “Distributed speech synthesis for embedded systems - an analysis,” in National Conference on Communication, Kharagpur, India, Jan 2005, pp 273-276.

2 M. Nageshwara Rao, Samuel Thomas, T. Nagarajan and Hema A. Murthy, “Text-to-speech synthesis using syllable-like units,” in National Conference on Communication, Kharagpur, India, Jan 2005, pp 277-280.

3 Samuel Thomas, M. Nageshwara Rao, Hema A. Murthy and C.S. Ramalingam, “Natural sounding TTS based on syllable-like units,” to appear in the proceedings of the 14th European Signal Processing Conference, Florence, Italy, Sep 2006.