SPEECH SYNTHESIS BY WORD CONCATENATION

by

Christos C. Vosnidis

A thesis submitted in partial fulfillment
of the requirements for the degree of

B.Sc. in Electronics and Computer Engineering

Technical University of Crete

2001

Supervisory Committee

Prof. Digalakis Vassilios (Supervisor)

Asoc. Prof. Dollas Apostolos

Prof. Paterakis Michael

Technical University of Crete

Abstract

SPEECH SYNTHESIS BY WORD CONCATENATION

by Christos C. Vosnidis

Chairperson of the Supervisory Committee:
Professor Vassilios Digalakis

Department of Electronics and Computer Engineering

The Weather Report Synthesizer is a speech synthesis system for weather forecasts in Greek. Instead of trying to improve the synthesis quality of PSOLA based concatenative speech synthesizers, we have chosen to use *words* as the synthesis unit for our system. This approach has the advantage of low complexity and quick implementation, while at the same time it achieves better speech quality due to the fact that the synthesis units inherently possess the necessary prosodic feature diversity. The selection of the optimal sequence of words that form the synthesized speech, however, presents the greatest challenge in the synthesis process. Several features are taken into consideration during the selection, but we have identified Coarticulation at the edges of consecutive words to have the greatest effect on the quality of the synthesized utterance. In this thesis we have developed a novel method for evaluating a measure on the coarticulation effects among pairs of words, based on feature clustering information as obtained from a current Speech Recognition System. The synthesizer's output was subject to a quality assessment procedure, the results of which are also presented.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Vassilis Digalakis for his guidance during the design and implementation of this application and his assistance in the preparation of this manuscript. In addition, special thanks to Professor Michael Paterakis, both for his help during the data acquisition process, as he had personally vouched for the author to the National Meteorological Agency (EMY), and for his remarks on the subject, as he was a member of the Supervisory. Finally, I would also like to thank Associate Professor Apostolos Dollas for the evaluation of this work, and for participating in the Supervisory Committee.

I would also like to thank the members of the evaluation team: Alexandraki A., Diplaris S., Giakoumis D., Georgiades I., Gorogia Th., Hajichrisafis N., Harizakis C., Kostoulakis C., Melessanaki E., Oikonomides D., Papadaki Z., Perakakis M., Pratsolis D., Revithi A., Salappa A., Stellakis D. and Tsourakis N. for their valuable help with extracting a measure on the quality of the output of this work.

And last, but not least, special thanks to Antonis Alygizakis, who has contributed his voice for the implementation of this system.

GLOSSARY

Corpus. The set of sentences to be recorded and later segmented into the words that they contain.

Concordance (of a file). A listing that contains all the words of that file, along with the line number on which the word occurs.

Reduction Property. A phonological word is **reduced** if it deviates from a canonical form a native speaker would judge as an acceptable version if the word were spoken in isolation.

Representative Sentence. A sentence that, when added to the collection of sentences already chosen to form the Corpus, introduces as much new information, in the form of words, either new or ones found in a context that has not yet been observed, as possible.

Shortest Path. The path with minimal cost is called shortest path.

Word (see also word class). A specific orthographic, case sensitive, word instance taking into consideration its association with any punctuation marks.

Word Class. The orthographic form of a word and its associated description.

Word Instance. A recorded word and its concrete description.

Word Layer. All word instances that belong to the same word class.

OVERVIEW

Chapter 1 provides a definition of the field of Computer Speech Synthesis, as well as a categorization based on the types of input to speech synthesizers, and on the basic methods used for the synthesis procedure.

Chapter 2 outlines the parts of a speech synthesizer, and provides insight to the current approaches in the construction of speech synthesizers.

Chapter 3 explains the basic idea of our approach to the subject, and outlines the strategy that our application uses to synthesize speech.

Chapter 4 provides explanation on the technical issues that have arisen during the implementation of the synthesizer, and exposes the inner workings of our speech synthesis system.

Finally, **Chapter 5** outlines our approach towards the definition of an evaluation procedure for the results of our application, and presents the findings of this overview.

The **Appendices** at the end of the thesis provide information on certain technical issues that may prove useful for some readers.

Chapter 1

COMPUTER SPEECH SYNTHESIS

Computer speech synthesis has reached a high level of performance, with increasingly sophisticated models of linguistic structure, low error rates in text analysis, and high intelligibility in synthesis from phonemic input. Mass-market applications have already been introduced, although the results are still not good enough for the ubiquitous application that such technology will eventually have.

A number of alternative directions of current research aim at the ultimate goal of fully natural synthetic speech. One especially promising trend is the systematic optimization of large synthesis systems, with respect to formal criteria of evaluation. Speech recognition has advanced rapidly in late '80s and early '90s through such approaches, and it seems likely that their application in synthesis will produce similar improvements.

1. WHAT IS COMPUTER SPEECH SYNTHESIS?

Let us begin this short review of Computer Speech Synthesis by first exploring what is meant by this term.

Obviously, this term refers to the creation by computer of human-like speech, but that only tells us what the output of the process is. Synthesized speech output may come from a wide range of processes that differ enormously in the nature of their inputs and the nature of their internal structures and calculations.

1.1. Types of Input

The input to a speech synthesizer may be

1. an uninterpreted reference to a previously recorded utterance

2. a message drawn from a small finite class of texts, such as telephone numbers
3. a message drawn from a larger, or even infinite, but still restricted, class of texts, such as names and addresses
4. a message drawn from unrestricted digital text, including anything from electronic mail to online newspapers to patent or legal texts, novels, or cookbooks
5. a message composed automatically from non-textual computer data structures (which we might thing of as analogous to "concepts" or "meanings"
6. a specification of the phonological content of a message, which for most applications must be produced from one of these types of input given previously

Most commercial applications so far have been of type 1 or 2. Classical "text-to-speech" systems are of type 4 and/or 6. Ultimate human computer interaction systems are likely to be of type 5, with a bit of 4.

A large number of the people involved in applying speech synthesis technology think that the most promising current opportunities are of type 3. Note that choosing such restricted domain applications has been crucial to the success of computer speech recognition. Most practical speech synthesis implementations, including our application, belong to this category.

1.2. Basic Methods

The system internal structures and processes of "speech synthesis" may involve

1. reproduction of digitally stored human voice, perhaps with compression/expansion
2. construction of messages by concatenation of digitally stored voice fragments
3. construction of messages by concatenation of digitally stored voice fragments, with modifications of the original timing and pitch
4. construction of messages by concatenation of digitally stored voice fragments, with rule-generated synthetic speech contours and rule generated segmental timing values
5. construction of messages using rule-generated synthetic time functions of acoustic parameters

6. construction of messages using rule-generated controls for the kinematics of simplified analogs of human vocal tract
7. construction of messages by realistic modeling of the physiological and physical processes of human speech production, including dynamic control of articulation and models of the airflow dynamics in the vocal tract

The largest scale of commercial activity has been of types 1 and 2, which might be called stored voice. This includes telecommunication intercepts, voice-mail prompts, and so forth. Much classical speech synthesis research has been of type 5 or 6, using techniques called *formant synthesis*. Several of the best current systems, and the most active areas of research, are of types 3 and 4, techniques that are called *concatenative synthesis*.

2. SPEECH SYNTHESIS BY WORD CONCATENATION

2.1. Overview

This thesis describes the concept and provides information on the implementation of a text-to-speech synthesizer that synthesizes sentences in the domain of whether forecasts by concatenating digitally stored voice fragments.

Each fragment corresponds to a specific word, and there is at least one fragment – usually three or more – corresponding to any given word. Our application uses a combination of criteria in order to select the optimal sequence of fragments to be used, applies, whenever necessary, basic energy smoothing at the borders of the fragments and concatenates them into the final synthetic utterance, in correspondence to the input text.

2.2. Categorization

Using the categorization described previously, and with respect to the type of input, our application can be assigned to the third category. We believe that there are certain applications for which speech synthesis is needed and where the implementation of a speech synthesizer with good speech quality is more important than that of a general-

purpose, open vocabulary synthesizer, with significant deterioration in speech quality. After all, most commercial applications of speech synthesis technologies involve the generation of speech from restricted, even though large enough, classes of text.

With respect to the basic methods used for the synthesis of the output speech, our application bares most resemblance to the second category, while using some new approaches to the selection process. It is certainly a concatenative synthesizer, where digitally stored words are used as the synthesis units, but which also uses techniques such as coarticulation matching and prominence in order to achieve better speech quality.

Chapter 2

TEXT-TO-SPEECH SYNTHESIS

1. INTRODUCTION

A Text-To-Speech (TTS) synthesizer is a computer-based system that should be able to read *any* text aloud, whether it was directly introduced in the computer by an operator or scanned and submitted to an Optical Character Recognition (OCR) system. Let us try to be clear. There is a fundamental difference between the system we are about to discuss here and any other talking machine (as a cassette-player for example) in the sense that we are interested in the automatic production of **new** sentences.

At first sight, this task does not look too hard to perform. After all, is not the human being potentially able to correctly pronounce an unknown sentence, even from his childhood? We all have, mainly unconsciously, a deep knowledge of the reading rules of our mother tongue. They were transmitted to us, in a simplified form, at primary school, and we improved them year after year. However, it would be a bold claim indeed to say that it is only a short step before the computer is likely to equal the human being in that respect. Despite the present state of our knowledge and techniques and the progress recently accomplished in the fields of Signal Processing and Artificial Intelligence, we would have to express some reservations. As a matter of fact, the reading process draws from the furthest depths, often unthought-of, of the human intelligence.

2. AUTOMATIC READING: WHAT FOR?

Each and every synthesizer is the result of a particular and original imitation of the human reading capability, submitted to technological and imaginative constraints that

are characteristic of the time of its creation. The concept of *high quality TTS synthesis* appeared in the mid eighties, as a result of important developments in speech synthesis and natural language processing techniques, mostly due to the emergence of new technologies (Digital Signal and Logical Inference Processors). It is now a must for the speech products family expansion.

Potential applications of High Quality TTS Systems are indeed numerous. Here are some examples:

- Telecommunications services.

  TTS systems make it possible to access textual information over the telephone. Knowing that about 70% of the telephone calls actually require very little interactivity, such a prospect is worth being considered. Texts might range from simple messages, such as local cultural events not to miss (cinemas, theatres, ...) , to huge databases which can hardly be read and stored as digitized speech. Queries to such information retrieval systems could be put through the user's voice (with the help of a speech recognition system), or through the telephone keyboard (with DTMF systems). One could even imagine that our (artificially) intelligent machines could speed up the query when needed, by providing lists of keywords, or even summaries. VoiceXML™ is a programming language designed for creating applications that enable access over the phone to information already available through a classic web browser. Using a server resident Voice Browser, voice and/or the telephone keypad as the method of data input, and server side speech synthesis as the method of data output, information services already offered on the Web could easily be modified to support mobile users. Given the continuously expanding number of mobile phones (more than 1B mobile phones worldwide) and the current trend in mobile Personal Digital Assistants, it seems that voice could easily be the medium of choice to address the needs of this new type of information services' users.

- Language education.

High Quality TTS synthesis can be coupled with a Computer Aided Learning system, and provide a helpful tool to learn a new language. To our knowledge, this has not been done yet, given the relatively poor quality available with commercial systems, as opposed to the critical requirements of such tasks.

- Aid to handicapped persons.

  Voice handicaps originate in mental or motor/sensation disorders. Machines can be an invaluable support in the latter case: with the help of an especially designed keyboard and a fast sentence assembling program, synthetic speech can be produced in a few seconds to remedy these impediments. Astrophysician Stephen Hawking gives all his lectures in this way. The aforementioned Telephone Relay Service is another example. Blind people also widely benefit from TTS systems, when coupled with Optical Recognition Systems (OCR), which give them access to written information. Mass-market synthesizers bundled with sound cards will soon invade the market for speech synthesis for blind users of personal computers. DECtalk™ is already available with the latest SoundBlaster™ cards now, although not yet in a form useful for blind people.

- Talking books and toys.

  The toy market has already been touched by speech synthesis. Many speaking toys have appeared, under the impulse of the innovative 'Magic Spell' from Texas Instruments. The poor quality available inevitably restrains the educational ambition of such products. High Quality synthesis at affordable prices might well change this.

- Vocal Monitoring.

  In some cases, oral information is more efficient than written messages. The appeal is stronger, while the attention may still focus on other visual sources of information. Hence the idea of incorporating speech synthesizers in measurement or control systems.

- Multimedia, man-machine communication.

In the long run, the development of high quality TTS systems is a necessary step (as is the enhancement of speech recognizers) towards more complete means of communication between men and computers. Multimedia is a first but promising move in this direction.

- Fundamental and applied research.

  TTS synthesizers possess a very peculiar feature, which makes them wonderful laboratory tools for linguists: they are completely under control, so that repeated experiences provide identical results (as is hardly the case with human beings). Consequently, they allow investigating the efficiency of intonative and rhythmic models. A particular type of TTS systems, which are based on a description of the vocal tract through its resonant frequencies (its *formants*) and denoted as *formant synthesizers*, has also been extensively used by phoneticians to study speech in terms of acoustical rules. In this manner, for instance, articulatory constraints have been enlightened and formally described.

3. HOW DOES A MACHINE READ?

It is tempting to think of the problem of converting written text into speech as "speech recognition in reverse". Current speech recognition systems are generally deemed successful if they can convert speech input into the sequence of words that was uttered by the speaker, so one might imagine that a TTS synthesizer would start with the words in the text, convert each word one into speech (being careful to pronounce each word correctly) and concatenate that result together.

However, when one considers what literate native speakers of a language must do when they read a text aloud, it quickly becomes clear that things are much more complicated than this simplistic approach suggests. Pronouncing words correctly is only part of the problem faced by human readers: in order to sound natural and to sound as if they understand what they are reading, they must also appropriately emphasize (accent) some words, and de-emphasize others; they must "chunk" the sentence into meaningful (intonational) phrases; they must pick an appropriate F_0

(fundamental frequency) contour; they must control certain aspects of their voice quality; they must know that a word should be pronounced longer if it appears in some positions in the sentence, than if it appears in others, since *segmental durations* are affected by various factors, including phrasal positions.

What makes reading such a difficult task, is that all writing systems systematically fail to specify many kinds of information that are important in speech. While the written form of a sentence (usually) completely specifies the words that are present, it will only partly specify the intonational phrases (typically with some form of punctuation), will usually not indicate which words to accent or de-accent, and hardly ever gives information on segmental duration, voice quality or intonation. One might think that a question mark '?' indicates that a sentence should be pronounced with a rising intonation: generally though a question mark merely indicates that a sentence is a question, leaving it up to the reader to judge whether this question should be rendered with a rising intonation. The orthographies of some languages – for instance Chinese, Japanese, and Thai – fail to give information on where word boundaries are, so that even this needs to be figured out by the reader.

The task of a TTS system is thus a complex one that involves mimicking what human readers do. But a machine is hobbled by the fact that it generally "knows" the grammatical facts of the language only imperfectly, and generally can be said to "understand" nothing of what it is reading. TTS algorithms thus have to do the best they can, making use, whenever possible, of purely grammatical information to decide on such things as accentuation, phrasing, and intonation, and coming up with a reasonable "middle ground" analysis for aspects of the output that are more dependent on actual understanding.

It is natural to divide the TTS problem into two broad sub-problems. The first of these is the conversion of text – an imperfect representation of language, as we have seen – into some form of linguistic representation, which includes information on the phonemes (sounds) to be produced, their duration, the locations and durations of any pauses and the F_0 contour to be used. The second – the actual synthesis of speech – takes this information and converts it into a speech waveform.

Figure 1 introduces the functional diagram of a very general TTS synthesizer. As for human reading, it comprises a Natural Language Processing module (NLP), capable of producing a phonetic transcription of the text read, together with the desired intonation and rhythm (often termed as *prosody*), and a Digital Signal Processing module (DSP), which transforms the symbolic information it receives into speech.

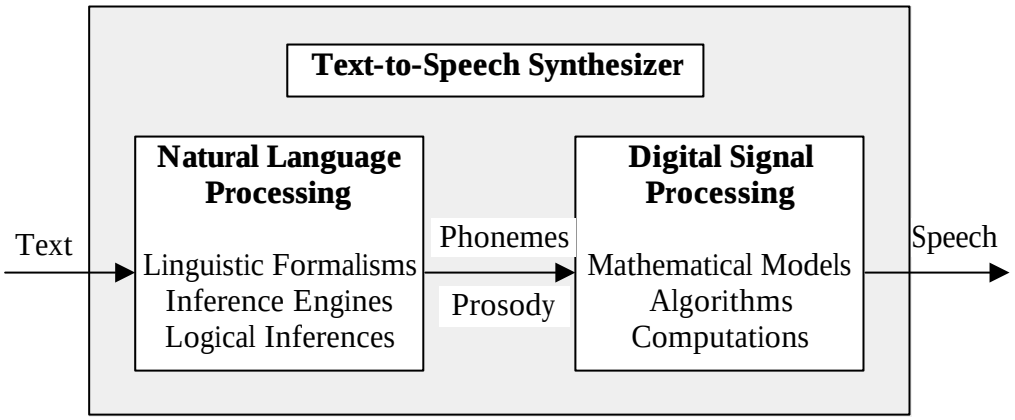

Figure 1 Functional Diagram of a TTS System

3.1. The NLP component

The Natural Language Processing block of a synthesizer is used to perform text and linguistic analysis on the input text, and can be broken down to the following parts:

- *Text Preprocessing*: including end-of-sentence detection, "text normalization" (expansion of numerals and abbreviations), and limited grammatical analysis, such as grammatical part-of-speech assignment.
- *Word Pronunciation*: including the pronunciation of names and the disambiguation of homographs.
- *Accent Assignment*: the assignment of levels of prominence to various words in the sentence.
- *Intonational Phrasing*: the breaking of (usually long) stretches of text into one or more intonational units.
- *Segmental Durations*: the determination, on the basis of linguistic information computed thus far, of appropriate durations for phonemes in the input.

- *F_0 contour computation.*

3.1.1. Text Preprocessing

The text preprocessing step performs the following tasks:

- It organizes the input sentences into manageable lists of words. It identifies numbers, abbreviations, acronyms and idiomatics and transforms them into full text when needed. An important problem is encountered as soon as the character level: that of punctuation ambiguity[1] (including the critical case of sentence end detection). It can be solved, to some extent, with elementary regular grammars.
- It performs a morphological analysis on the input text, in order to propose all possible part of speech categories for each word taken individually, on the basis of their spelling. Inflected, derived, and compound words are decomposed into their elementary graphemic units (their *morphs*) by simple regular grammars exploiting lexicons of stems and affixes.
- Finally words are considered in their context, which allows for the reduction of the list of their possible part of speech categories to a very restricted number of highly probable hypotheses, given the corresponding possible parts of speech of neighboring words. This can be achieved either with *n-grams*, which describe local syntactic dependences in the form of probabilistic finite state automata (i.e. as a Markov model), to a lesser extent with *multi-layer perceptrons* (i.e., neural networks) trained to uncover contextual rewrite rules, or with *local, non-stochastic grammars* provided by expert linguists or automatically inferred from a training data set with *classification and regression tree* (CART) techniques.
- Finally, a syntactic-prosodic parser, which examines the remaining search space and finds the text structure (i.e. its organization into clause and phrase-like constituents) which more closely relates to its expected prosodic realization (see below).

[1] A period '.' may usually used as a sentence delimiter, may also be used for instance to mark abbreviations.

3.1.2. Word Pronunciation

The Word Pronunciation module is responsible for the automatic determination of the phonetic transcription of the incoming text. It thus seems, at first sight, that its task is as simple as performing the equivalent of a dictionary look-up! From a deeper examination, however, one quickly realizes that most words appear in genuine speech with several phonetic transcriptions, many of which are not even mentioned in pronunciation dictionaries. Namely:

1. Pronunciation dictionaries refer to word roots only. They do not explicitly account for morphological variations (i.e. plural, feminine, conjugations, especially for highly inflected languages, such as French), which therefore have to be dealt with by a specific component of phonology, called morphophonology.
2. Some words actually correspond to several entries in the dictionary, or more generally to several morphological analyses, generally with different pronunciations. This is typically the case of heterophonic homographs, i.e. words that are pronounced differently even though they have the same spelling, as for 'record' (/rek • ùd/ or /rIk • ùd/), constitute by far the most tedious class of pronunciation ambiguities. Their correct pronunciation generally depends on their part-of-speech and most frequently contrasts verbs and non-verbs, as for 'contrast' (verb/noun) or 'intimate' (verb/adjective), although it may also be based on syntactic features, as for 'read' (present/past)
3. Pronunciation dictionaries merely provide something that is closer to a phonemic transcription than from a phonetic one (i.e. they refer to phonemes rather than to phones). Consonants, for example, may reduce or delete in clusters, a phenomenon termed as consonant cluster simplification, as in 'softness' [s • fnIs] in which [t] fuses in a single gesture with the following [n].
4. Words embedded into sentences are not pronounced as if they were isolated. Surprisingly enough, the difference does not only originate in variations at word boundaries (as with phonetic liaisons), but also on alternations based on the organization of the sentence into non-lexical units, that is whether into groups of words (as for phonetic lengthening) or into non-lexical parts thereof (many phonological processes, for instance, are sensitive to syllable structure).

5. Finally, not all words can be found in a phonetic dictionary: the pronunciation of new words and of many proper names has to be deduced from the one of already known words.

Clearly, points 1 and 2 heavily rely on a preliminary morphosyntactic (and possibly semantic) analysis of the sentences to read. To a lesser extent, it also happens to be the case for point 3 as well, since reduction processes are not only a matter of context-sensitive phonation, but they also rely on morphological structure and on word grouping, that is on morphosyntax. Point 4 puts a strong demand on sentence analysis, whether syntactic or metrical, and point 5 can be partially solved by addressing morphology and/or by finding graphemic analogies between words.

3.1.3. Prosody Generation

The term *prosody* refers to certain properties of the speech signal, which are related to audible changes in pitch, loudness, and syllable length. Prosodic features have specific functions in speech communication (see Figure 2). The most apparent effect of prosody is that of focus. For instance, there are certain pitch events which make a syllable stand out within the utterance, and indirectly the word or syntactic group it belongs to will be highlighted as an important or new component in the meaning of that utterance. The presence of a focus marking may have various effects, such as contrast, depending on the place where it occurs, or the semantic context of the utterance.

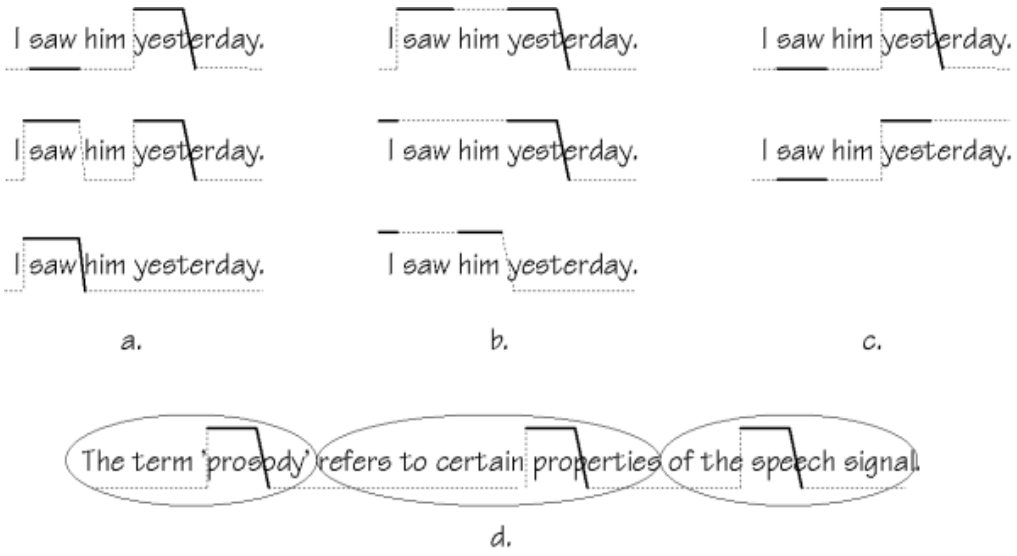

Figure 2 Different kinds of information provided by intonation (lines indicate pitch movements; solid lines indicate stress).
a. Focus or given/new information;
b. Relationships between words (saw -yesterday; I-yesterday; I-him)
c. Finality (top) or continuation (bottom), as it appears on the last syllable;
d. Segmentation of the sentence into groups of syllables.

Although maybe less obvious, there are other, more systematic or general functions.

Prosodic features create a segmentation of the speech chain into groups of syllables, or, put the other way round, they give rise to the grouping of syllables and words into larger chunks. Moreover, there are prosodic features, which indicate relationships between such groups, indicating that two or more groups of syllables are linked in some way. This grouping effect is hierarchical, although not necessarily identical to the syntactic structuring of the utterance.

Accentuation

Various words in a sentence are associated with accents, which are usually manifested as upward or downward movements of fundamental frequency. Accentuation, along with intonational phrasing and F_0 contour computation is part of the large problem of prosody generation.

Back to accentuation, words are typically distinguished into three groups with regard to their *prominence*. Two are *accented* and *unaccented*, and the third is *cliticized*. Cliticized words are unaccented but in addition have lost their word stress, so that they tend to be short in duration: in effect they behave like unstressed affixes, even though they are written as separate words.

Accents are assigned primarily on the basis of broad lexical categories or parts of speech. Content words – nouns, verbs, and adjectives tend in general to be accented; function words, including auxiliary verbs and propositions tend to be unaccented; short function words tend to be cliticized. However, more complex accentuation schemes based on syntactic and semantic analysis have also been used, providing better results.

Intonational Phrasing

Most commercially developed TTS system have emphasized coverage rather than linguistic sophistication, by concentrating their efforts on text analysis strategies aimed to segment the *surface structure* of incoming sentences, as opposed to their syntactically, semantically, and pragmatically related *deep structure*. The resulting syntactic-prosodic descriptions organize sentences in terms of prosodic groups strongly related to phrases (and therefore also termed as *minor* or *intermediate phrases*), but with a very limited amount of embedding, typically a single level for these minor phrases as parts of higher-order prosodic phrases (also termed as *major* or *intonational phrases*, which can be seen as a prosodic-syntactic equivalent for clauses) and a second one for these major phrases as parts of sentences, to the extent that the related major phrase boundaries can be safely obtained from relatively simple text analysis methods. In other words, they focus on obtaining an acceptable segmentation and translate it into the continuation or finality marks of Figure 2.c , but ignore the relationships or contrastive meaning of Figure 2.a and b.

Liberman and Church, for instance, have reported on such a very crude algorithm, termed as the *chinks 'n chunks* algorithm, in which prosodic phrases (which they call *f-groups*) are accounted for by the simple regular rule:

a (minor) prosodic phrase = a sequence of chinks followed by a sequence of chunks

in which *chinks* and *chunks* belong to sets of words which basically correspond to function and content words, respectively, with the difference that objective pronouns (like '*him*' or '*them*') are seen as chunks and that tensed verb forms (such as '*produced*') are considered as chinks. They show that this approach produces efficient grouping in most cases, slightly better actually than the simpler decomposition into sequences of function and content words, as shown in the example below:

function words / content words	**chinks / chunks**
I asked	*I asked them*
them if they were going home	*if they were going home*
to Idaho	*to Idaho*
and they said yes	*and they said yes*
and anticipated	*and anticipated one more stop*
one more stop	*before getting home*
before getting home	

Other, more sophisticated approaches include syntax-based expert systems, and automatic, corpus-based methods as with the *classification and regression tree* (CART) techniques.

Segmental Durations

Once the phonemes to be produced by the synthesizer have been computed, it is necessary to decide how long to make each one. What duration to assign to a phonemic segment depends upon many factors, including:

- The identity of the segment in question.
- The stress of the syllable of which the segment is a member
- Whether the syllable of which the segment is a member bears an accent.
- The quality of the surrounding segments.
- The position of the segment in the phrase.

Some methods involve the use of *duration rules*, which are rules of the form "if the segment is X and it is in phrase-final position, then lengthen X by n msec". These rules can be formalized explicitly in terms of *duration models*, which are mathematical expressions prescribing how the various conditioning factors are to be used in computing the durations of segments. We could even use exploratory data analysis, to arrive to models whose predictions show a good fit to durations from a corpus of labeled speech.

Sentence Intonation

Information such as:

- The syllables in the utterance to be stressed, as computed by the accentuation and the pronunciation module.
- The type of accents to be used, as well as the types of initial and final boundary tones and phrase accents.
- The duration of the segments in the utterance.

Sentence intonation is implemented by the F_0 contour of the phrase. However, its generation is not straightforward either. It requires formalizing a lot of phonetic or phonological knowledge, either obtained from experts or automatically acquired from data with statistical methods. More information on this can be found in [Dutoit].

3.2. The DSP component

Once the text has been transformed into phonemes, and their associated durations and a fundamental frequency contour have been computed, the system is ready to compute the speech parameters for synthesis.

Intuitively, the operations involved in the DSP module are the computer analogue of dynamically controlling the articulatory muscles and the vibratory frequency of the vocal folds so that the output signal matches the input requirements. In order to do it properly, the DSP module should obviously, in some way, take articulatory constraints into account, since it has been known for a long time that phonetic

transitions are more important than stable states for the understanding of speech. This, in turn, can be basically achieved in two ways:

- Explicitly, in the form of a series of rules which formally describe the influence of phonemes on one another;
- Implicitly, by storing examples of phonetic transitions and co-articulations into a speech segment database, and using them just as they are, as ultimate acoustic units (i.e. in place of phonemes).

Two main classes of TTS systems have emerged from this alternative, which quickly turned into synthesis philosophies given the divergences they present in their means and objectives: *synthesis-by-rule* and *synthesis-by-concatenation*.

3.2.1. Rule-based synthesizers

Rule-based synthesizers are mostly in favor with phoneticians and phonologists, as they constitute a cognitive, generative approach of the phonation mechanism. Rule-based approaches are space-efficient, since they eliminate the need to store speech segments, and they also make it easier in principle to implement new speaker characteristics for different voices, as well as different phone inventories for new dialects and languages.

These systems are also restrictive regarding the choice of the parametric representation of speech, since such schemes rely both on our understanding of the relation between the parameters and the acoustic signals they represent, and on our ability to compute the dynamics of the parameters as they move from one sound to another. Thus far only articulation parameters and formants have been used in rule-based systems.

Most such systems describe speech as the dynamic evolution of up to 60 parameters, mostly related to formant and anti-formant frequencies and bandwidths together with glottal waveforms. Clearly, the large number of (coupled) parameters complicates the analysis stage and tends to produce analysis errors. What is more, formant frequencies and bandwidths are inherently difficult to estimate from speech data. The need for

intensive trials and errors, in order to cope with analysis errors, makes them time-consuming systems to develop (several years are commonplace). Yet, the synthesis quality achieved up to now reveals typical buzzyness problems, which originate from the rules themselves: introducing a high degree of naturalness is theoretically possible, but the rules to do so are still to be discovered.

Rule-based synthesizers remain, however, a potentially powerful approach to speech synthesis. They allow, for instance, to study speaker-dependent voice features so that switching from one synthetic voice into another can be achieved with the help of specialized rules in the rule database. Following the same idea, synthesis-by-rule seems to be a natural way of handling the articulatory aspects of changes in speaking styles (as opposed to their prosodic counterpart, which can be accounted for by concatenation-based synthesizers as well). No wonder then that it has been widely integrated into TTS systems (MITalk™ and the JSRU synthesizer for English).

3.2.2. Concatenative synthesizers

As opposed to rule-based ones, *concatenative synthesizers* possess a very limited knowledge of the data they handle: most of it is embedded in the segments to be chained up. This clearly appears in Figure 3, where all the operations that could indifferently be used in the context of a music synthesizer (i.e. without any explicit reference to the inner nature of the sounds to be processed) have been grouped into a *sound processing* block, as opposed to the upper *speech processing* block whose design requires at least some understanding of phonetics.

Database preparation

A series of preliminary stages have to be fulfilled before the synthesizer can produce its first utterance. At first, segments are chosen so as to minimize future concatenation problems. A combination of diphones (i.e. units that begin in the middle of the stable state of a phone and end in the middle of the following one), half-syllables, and triphones (which differ from diphones in that they include a complete central phone) are often chosen as speech units, since they involve most of the transitions and coarticulations while requiring an affordable amount of memory.

When a complete list of segments has emerged, a corresponding list of words is carefully completed, in such a way that each segment appears at least once (twice is better, for security). Unfavorable positions like inside stressed syllables or in strongly reduced (i.e. over-coarticulated) contexts, are excluded. A corpus is then digitally recorded and stored, and the elected segments are spotted, either manually with the help of signal visualization tools, or automatically thanks to segmentation algorithms, the decisions of which are checked and corrected interactively. A segment database finally centralizes the results, in the form of the segment names, waveforms, durations, and internal sub-splittings. In the case of diphones, for example, the position of the border between phones should be stored, so as to be able to modify the duration of one half-phone without affecting the length of the other one.

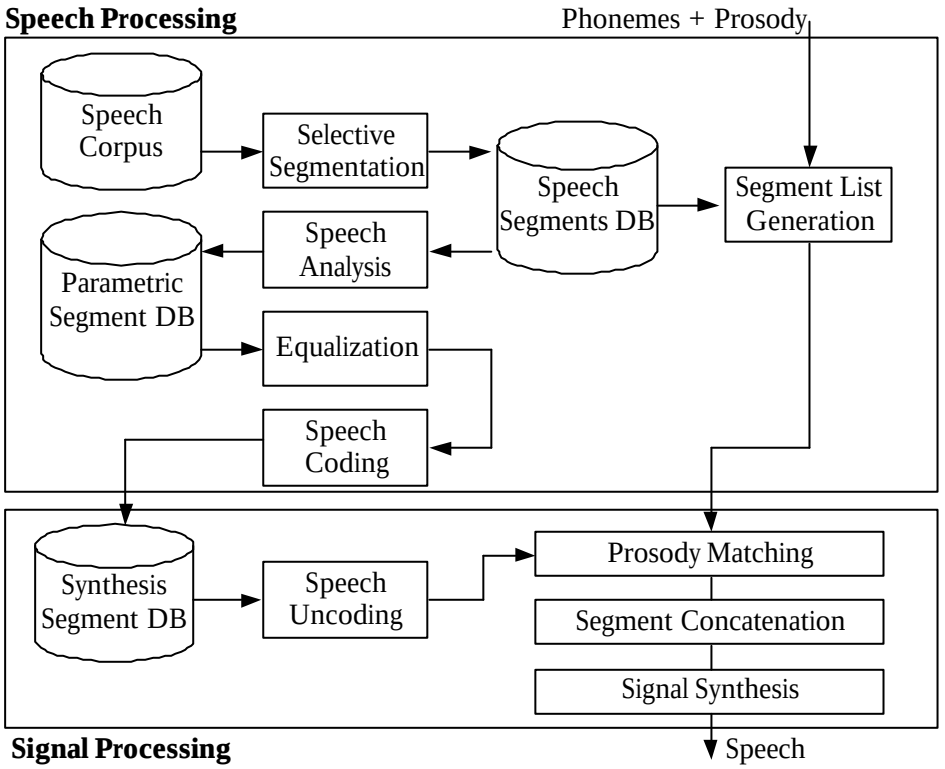

Figure 3 A general concatenation-based synthesizer. The upper left hatched block corresponds to the development of the synthesizer (i.e. it is processed once for all). Other blocks correspond to run-time operations. A flag indicates language-dependent operations and data.

Segments are then often given a parametric form, in the form of a temporal sequence of vectors of parameters collected at the output of a *speech analyzer* and stored in a

parametric segment database. The advantage of using a speech model originates in the fact that:

- Well chosen speech models allow data size reduction, an advantage which is hardly negligible in the context of concatenation-based synthesis given the amount of data to be stored. Consequently, a parametric speech coder often follows the analyzer.
- A number of models explicitly separate the contributions of respectively the source and the vocal tract, an operation that remains helpful for the pre-synthesis operations: prosody matching and segments concatenation.

Indeed, the actual task of the synthesizer is to produce, in real-time, an adequate sequence of concatenated segments, extracted from its parametric segment database and the prosody of which has been adjusted from their stored value, i.e. the intonation and the duration they appeared with in the original speech corpus, to the one imposed by the language processing module. Consequently, the respective parts played by the prosody matching and segments concatenation modules are considerably alleviated when input segments are presented in a form that allows easy modification of their pitch, duration, and spectral envelope, as is hardly the case with crude waveform samples.

Since segments to be chained up have generally been extracted from different words, i.e. in different phonetic contexts, they often present amplitude and timbre mismatches. Even in the case of stationary vocalic sounds, for instance, a rough sequencing of parameters typically leads to audible discontinuities. These can be coped with during the constitution of the synthesis segments database, thanks to an *equalization* in which related endings of segments are imposed similar amplitude spectra, the difference being distributed on their neighborhood. In practice, however, this operation is restricted to amplitude parameters: the equalization stage smoothly modifies the energy levels at the beginning and at the end of segments, in such a way as to eliminate amplitude mismatches (by setting the energy of all the phones of a given phoneme to their average value). In contrast, timbre conflicts are better tackled at run-time, by *smoothing* individual couples of segments when necessary rather than

equalizing them once for all, so that some of the phonetic variability naturally introduced by co-articulation is still maintained. In practice, amplitude equalization can be performed either before or after speech analysis (i.e. on crude samples or on speech parameters).

Once the parametric segment database has been completed, synthesis itself can begin.

Speech synthesis

A sequence of segments is first deduced from the phonemic input of the synthesizer, in a block termed as *segment list generation* in Figure 3, which interfaces the NLP and DSP modules. Once prosodic events have been correctly assigned to individual segments, the *prosody matching* module queries the synthesis segment database for the actual parameters, adequately encoded, of the elementary sounds to be used, and adapts them one by one to the required prosody. The segment concatenation block is then in charge of dynamically matching segments to one another, by smoothing discontinuities. Here again, an adequate modeling of speech is highly profitable, provided simple interpolation schemes performed on its parameters approximately correspond to smooth acoustical transitions between sounds. The resulting stream of parameters is finally presented at the input of a synthesis block, the exact counterpart of the analysis one. Its task is to produce speech.

Segmental quality

The efficiency of concatenative synthesizers to produce high quality speech is mainly subordinated to:

1. The type of segments chosen.

   Segments should obviously exhibit some basic properties:

   - They should account for as many co-articulatory effects as possible.
   - Given the restricted smoothing capabilities of the concatenation block, they should be easily connectable.
   - Their number and length should be kept as small as possible.

- On the other hand, longer units decrease the density of concatenation points, therefore providing better speech quality. Similarly, an obvious way of accounting for articulatory phenomena is to provide many variants for each phoneme. This is clearly in contradiction with the limited memory constraint. Some trade-off is necessary. Diphones are often chosen. They are not too numerous (about 1200 for French, including lots of phoneme sequences that are only encountered at word boundaries, for 3 minutes of speech, i.e. approximately 5 Mbytes of 16 bits samples at 16 kHz) and they do incorporate most phonetic transitions. No wonder then that they have been extensively used. They imply, however, a high density of concatenation points (one per phoneme), which reinforces the importance of an efficient concatenation algorithm. Besides, they can only partially account for the many co-articulatory effects of a spoken language, since these often affect a whole phone rather than just its right or left halves independently. Such effects are especially patent when somewhat transient phones, such as liquids and (worst of all) semi-vowels, are to be connected to each other. Hence the use of some larger units as well, such as triphones.

2. The model of speech signal, to which the analysis and synthesis algorithms refer.

The models used in the context of concatenative synthesis can be roughly classified into two groups, depending on their relationship with the actual phonation process. *Production models* provide mathematical substitutes for the part respectively played by vocal folds, nasal and vocal tracts, and by the lips radiation. Their most representative members are Linear Prediction Coding (LPC) synthesizers, and the formant synthesizers we mentioned in *Section 3.2.1. Rule-based synthesizers*. On the contrary, *phenomenological models* intentionally discard any reference to the human production mechanism. Among these pure digital signal processing tools, spectral and time-domain approaches are increasingly encountered in TTS systems. Two leading such models exist: the hybrid Harmonic/Stochastic (H/S) model of [Abrantes] and the Time-Domain Pitch-Synchronous-OveraLap-Add (TD-PSOLA) one [Moulines &

Charpentier]. The latter is a time-domain algorithm: it virtually uses no speech explicit speech model. It exhibits very interesting practical features: a very high speech quality (the best currently available) combined with a very low computational cost (7 operations per sample on the average). The hybrid Harmonic/stochastic model is intrinsically more powerful than the TD-PSOLA one, but it is also about ten times more computationally intensive. PSOLA synthesizers are now widely used in the speech synthesis community. The recently developed MBROLA algorithm [Dutoit] even provides a time-domain algorithm which exhibits the very efficient smoothing capabilities of the H/S model (for the spectral envelope mismatches that cannot be avoided at concatenation points) as well as its very high data compression ratios (up to 10 with almost no additional computational cost) while keeping the computational complexity of PSOLA.

Chapter 3

SYNTHESIS BY WORD CONCATENATION

1. INTRODUCTION

Common speech synthesis systems are based on predefined units, whose concatenation is obligatory. Small speech units such as diphones or demisyllables recorded from a human speaker are concatenated to build the synthetic utterance. The prosodic structure is modeled on the basis of artificial F_0-, energy-, and duration parameters, which are applied to the synthesis units in order to build the synthetic utterance. This results in synthetic speech for unrestricted domains, but the synthesized speech has a machine-like quality. However, in many cases, synthesis for unrestricted domains is not necessary, because those speech-synthesis applications operate on restricted domains.

Recent synthesis approaches are also based on the concatenation of recorded units, but the concatenation is not obligatory. In addition, instead of modeling prosodic parameters explicitly, the inherent prosodic structure of the recorded speech signals is used. This implies that the speech corpus contains each synthesis unit in different prosodic settings. Furthermore, a method to select the appropriate unit sequence to be synthesized is necessary. Usually the synthetic speech generated with these approaches is judged to be more natural than that from diphone synthesis.

Our method is based on the observation that an utterance sounds more natural when it is completely stored in the corpus. In that case, no concatenation of units is necessary, just a simple playback of the recorded utterance is sufficient. From this observation follows the fact that larger units yield better synthetic speech. But of course it is impossible to record all possible utterances for a specific task. For this, we have decided to use words as our basic synthesis units.

The Weather Forecast domain features approximately 400 words. The recording of each word in the domain in only one instance would have resulted in poor synthesis quality, because the pronunciation variations of words depending of their context are not modeled. To obtain words in their natural surroundings, a number of sentences are chosen from actual weather forecast transcriptions, where all needed words are included with sufficient variations. Those sentences are spoken by a human speaker and comprise our speech corpus.

At first sight, our method looks simple. But our problem is the following:

> *When is a recorded unit appropriate to be used at a given place in the synthetic utterance?*

We have observed that few criteria are sufficient to achieve close to naturally sounding speech synthesis. Additionally, the time for creating and annotating the corpus as well as computing cost for the selection algorithm is smaller than in approaches that use phonemes or other small synthesis units.

2. CORPUS CONSTRUCTION

2.1. Corpus Definition

The success of the speech synthesis schema outlined above, crucially depends on an effective corpus design, such that instances of all necessary units can be found in matching prosodic context.

The domain that the application is built to cover is limited, but still quite large when compared to the domains of other closed-vocabulary tasks, such as the synthesis of telephone numbers. Its difficulty lies in the fact that it involves the synthesis of whole sentences, rather than certain words within a sentence. However, there is some form of syntactic uniformity within that domain, since the definition of the domain was based on weather forecast reports produced by the National Meteorological Agency (EMY).

The majority of print- and electronic media use these forecasts, almost without any modification, to report on the near future's weather conditions. Having been produced by a single government agency, these reports follow common guidelines, regarding their syntax.

We will explain how we took advantage of this observation during the task of the corpus definition.

2.1.1. The Weather Forecast Report

The Weather Forecast, published daily by EMY, and used by the majority of print and electronic media with almost no modification at all, has the following form:

Ge????? pa?at???se??, p??e?d?p???se??.

<Ge???? ? a?a?t???st???>

Ge???? p?????s? ??a s?µe?a <? µ??a>

<?a????? F a ???µe?a>

<??tas? ??e????s? ???µ??>

Ge???? p?????s? ??a a???? <? µ??a>

<?a????? F a ???µe?a>

<??tas? ??e????s? ???µ??>

<Te?µ???as?a>

<? ?at?t?ta>

??p???? p?????se??.

????a.

G?a s?µe?a <? µ??a>

<?a????? F a ???µe?a>

<??tas? ??e????s? ???µ??>

G?a a???? <? µ??a>

< ?a ????? F a ???µe?a>

< ??tas? ??e????s? ???µ??>

<Te?µ???as?a>

Tessa??????.

G?a s?µe?a <? µ??a>

< ?a ????? F a ???µe?a>

< ??tas? ??e????s? ???µ??>

G?a a???? <? µ??a>

< ?a ????? F a ???µe?a>

< ??tas? ??e????s? ???µ??>

<Te?µ???as?a >

It is obvious that the report has an inherent structure, containing fields of common meaning and syntax that are found in several parts of the report. For example, the <Temperature> field can be found both in the <General Forecast> and the <Local Forecast> part of the report, and within the later, both in the <Athens Forecast> and the <Thessalonica Forecast> sub-parts. The syntax of the <Temperature> field is common to all its instances within the report. This means that we can use any of the four sentences describing the temperature and rearrange them randomly in their positions in the report, and still get a meaningful weather forecast.

From the schema that we are presenting above, we can associate the information that lies within the forecast with one of the following categories:

- General Characteristics
- Weather Phenomena
- Wind Direction – Intensity

- Temperature
- Visibility

We have used this observation in the selection process of the sentences that were to be used for the creation of the Corpus.

We began by assigning the sentences of the original data set into one of the aforementioned categories. Then, we used the sentences belonging to each category in order to determine that category's dictionary. For instance, the sentences that lie within the temperature category were extracted from the original data set and used to determine the vocabulary used to describe a day's temperature variations.

2.1.2. Selection of Sentences

The creation of the corpus, the set of sentences to be recorded and later segmented into the words that they contain, is crucial to the performance of our application. After all, the fragments extracted from this process are the basic units used to synthesize the output of our TTS system. It is the efficiency and the quality of the Corpus Creation procedure that largely defines the success of our application.

The sentences that were finally selected were chosen from a set of transcribed weather forecast reports, covering a week of each month during the period October 1999 – September 2000. These reports were kindly provided by the National Meteorological Agency (EMY). For these data, the following information is given:

Total Number of Sentences	1676
Total Number of Word Instances	19138
Total Number of Distinct Words	389

Table 1 Statistics for the Original Data Set

Corpus Characteristics

The following requirements were to be met by the corpus:

1. The corpus should not contain an excessive number of sentences.

Since only one human operator performs the segmentation and the phonetic annotation, the size of the corpus should be contained, as much as the quality of the output is not severely deteriorated.

2. The corpus must contain all the words that are used in our application.
   We need at least one recorded instance of all words that may be found in the application.
3. The corpus should contain these words in as many contexts as possible.
   Multiple recorded instances of commonly used words should be available to the application, in order to incorporate into the corpus as many prosodic features as possible.

Selection Strategy

In practice, we were facing the following problem:

> *Define a procedure for the selection of the most representative sentences to be used for the creation of the Corpus.*

With the term *representative* we wish to describe a sentence that, when added to the collection of sentences already chosen, introduces as much new information, in the form of words, either new or ones found in a context that has not yet been observed, as possible.

The strategy used for the selection procedure was simple:

> *Given the list of all distinct words that comprise the vocabulary of the Weather Reports, add into the Corpus the sentence that contains most of these words. After that, delete these words from the list, repeating the above procedure until the list is empty.*

Optimizations

The purpose for this procedure is the creation of a corpus of sentences that contain the same set of words as the original data set, and at the same time the instances of

these words exhibit such diversity that account for as many prosodic phenomena as possible.

We chose the sentences to add into the corpus rather not from the whole data set, but from the context specific subsets that we have identified during the analysis of the weather forecast reports (see also *Section 2.1.1. The Weather Forecast Report*). There are two reasons that have led us to this decision:

1. The dictionary size of each subset is considerably smaller than that of the whole data set.
2. Each subset has a different syntax, and the context in which a certain word can be observed, varies according to this syntax.

This splicing of the selection procedure further expands the size of the final corpus. However, since more intra-sentence phenomena are taken into account, we will finally be getting better quality for the synthesized speech.

Finally, we wanted every word to be available in the corpus in all its forms. This means that we consider each orthographic representation of the same word, to be a different word, e.g. "μ??a?" and "μ??a". In addition to that we also take into consideration both the relative position of the word in the sentence, and any punctuation marks. We do that since the pronunciation of that same word is different in each of the cases described above.

Word	Word Instances
xyz	xyz
	xyz,
	xyz.
	Xyz

Table 2 Words and "Words"

From now on, when referring to a *word* (and later *word class*) we will be referring to a specific orthographic, case sensitive word instance, taking into consideration its association with any punctuation marks. Using that definition, the four word instances found in the second column of Table 2 are considered as distinct *words*.

This categorization has the effect of further expanding the corpus's size, but with the effect of achieving better quality of the synthetic speech.

2.1.3. The final Corpus

As we have already stated, the original data set contained almost 20,000 words and some 1,700 sentences.

The selection procedure that we have described produced a corpus of sentences that were used for the extraction of the basic synthesis units of our application, and that has the following statistical characteristics.

	Original Data Set	Corpus Sentences
Distinct Words	521	521
Total Words	19138	2494
Instances/Word	36.73	4.79
Total Sentences	1676	163
Words/Sentence	11.42	15.30

Table 3 Statistical Analysis of the Corpus

An analysis of the words found in the corpus's sentences shows that each word has at least two instances. The majority of words (98%) have at least three instances, while there exist certain words, such as articles, conjunctions and key words that can be found at least 50 times.

2.2. Recording Phase

The speaker was instructed to read the sentences well articulated but as naturally as possible thus resulting in context-specific phonetic assimilation.

The sentences were recorded in laboratory environment, with low levels of noise, and were stored digitally using PCM coding at 16,000Hz with 16bits/sample. The corpus had a size of 34,066,196 bytes, equal to 1064.5 sec of speech.

2.3. Post-Recording Phase

The post-recording phase can be divided into the following four stages:

1. Quality check
2. Energy Normalization
3. Segmentation
4. Prosodic Labeling

Quality Check

In the post-recording phase, the material was object to auditive- and acoustic quality checks. Some sentences were recorded again, as they did not meet our quality standards regarding noise, articulation clarity, and sentence intonation.

Energy Normalization

Although the speaker had been instructed to read the Corpus's sentences without alterations in the volume of his voice among these sentences, early quality checks had shown that the volume among certain sentences had some fluctuation. Thus, all sentences where subjected to mean energy normalization. Using the `wavdynanorm` utility of the Nuance® v.7.0.2 Speech Recognition System, all the recorded sentences were submitted to DC offset removal and average energy normalization, with a target output level of -17.6 dBm.

Segmentation

After that, the corpus was segmented into words, both manually and automatically.

The automatic segmentation was made possible by using the `batchrec` utility of the Nuance® v.7.0.2 Speech Recognition System, in the `force-align` mode. `batchrec` was used to perform recognition on the set of the Corpus's Sentences, given the transcriptions of these sentences. Thus, an output file was produced, containing, among others, information on the exact locations of words and pauses in the recorded sentences. Using this alignment information, the sentences were segmented into the words they contain.

Prosodic Labeling

In the last stage of corpus construction, the material had to be labeled according to the prosodic criteria, which are later taken into account by the unit selection (see *Section 3.3. Selection Criteria*). Preliminary tests showed that the following labels influencing the prosodic and segmental form are necessary for a naturally sounding synthesis based upon word concatenation.

1. Utterance position (initial / rnedial / final)
2. Sentence modality (interrogative / declarative)
3. Reduction
4. Coarticulatory effects

Utterance Position

It has been found that words that are used in a position different than the position they are originally found in the corpus greatly deteriorate the quality of the synthesized sentence. This information was utilized to make sure that, whenever possible, a word should be used in the position that was originally found.

Position labeling is performed automatically by using the information provided in the name of each fragment (see *Segment Naming Policy* in *Section 3.1.1. Word Segment Database*).

Sentence Modality

Modality labeling, although supported by our application, was not necessary in the domain of Weather Forecasts, since all the sentences in the domain are declarative.

Reduction

For reduction labeling the following definition was used:

> *A phonological word is **reduced** if it deviates from a canonical form a native speaker would judge as an acceptable version if the word were spoken in isolation.*

This – certainly debatable – definition applies to the phenomena of both reduction, marked as some kind of *target undershoot,* and *contextual assimilation.*

In order to guarantee the comprehensibility of the synthesized utterance unreduced words are preferred by unit selection. An exception to this rule is only given if the word to be synthesized is available in a matching context. It is obvious that the reduction property for each word needed to be determined **auditively**, something that was actually done for all segments in the Corpus.

Coarticulatory Effects

Coarticulatory phenomena are due to the fact that each articulator moves continuously from the realization of one phone to the next. They appear even in the most careful speech. In our case we were interested only to the coarticulatory phenomena observed at the *edges of words*, since they affect the smoothness of the transitions from one word to the next, and thus need to be taken into account during the selection procedure.

Coarticulatory labeling consists of keeping, for each segment in the Corpus, the phonemes that are found at the edges of the word. This means that we are keeping the first and last phoneme of the considered word and the last and first phoneme of the words preceding and following it, respectively, in the considered sentence of the Corpus (for more information see *Section 3.3.1. Cost Functions*)

3. UNIT SELECTION

Our corpus contains identical words in different lexical or prosodic contexts. As in object oriented programming languages we will call the orthographic form of a word and its associated description a **class**, a recorded word and its concrete description an **instance**.

For each unit class given by the utterance description there exist several unit instances. All possible combinations of these instances, which will form the correct synthetic utterance, are potential solutions to our problem. We have to decide which

combination of unit instances is the best. We do this by evaluating a cost function for each unit combination. The solution we take is that sequence of units whose value of the cost function is minimal.

3.1. Using Graphs for Unit Selection

A formal way of minimizing the value of the cost function over all possible sequences of word instances for a given sentence is given by graph theory. After a short introduction to graphs, we will show that the problem of finding the best sequence of word instances to be used for the synthesis of a given sentence can be viewed as the problem of identifying the shortest path in a directed, weighted graph.

3.1.1. Definitions

A graph $G = (V, E)$ consists of a set of *vertices* V and *edges* E.

Each edge is a pair (v, w), where $v, w \in V$. If the pair is ordered, then the graph is called *directed*.

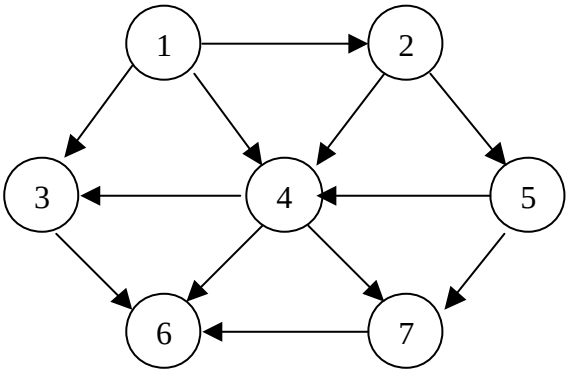

Figure 4 A directed graph

Vertex w is *adjacent* to v, if and only if $v, w \in E$. In an undirected graph with edge (v,w) and hence (w, v), v is adjacent to w and vice versa.

A *path* in a graph is a sequence of vertices $w_1, w_2, \ldots, w_N$ such that $(w_i, w_{i+1}) \in E$ for $1 \leq I < N$. The *length* of such a path is the number of edges on the path, which is equal

to N – 1. We allow a path from a vertex to itself; if this path contains no edges, then the path length is 0.

A *simple path* is a path such that all the vertices are distinct, except that the first and the last could be the same.

A *cycle* in a directed graph is a path of length at least 1, such that $w_1 = w_N$; this cycle is simple if the path is simple. For an undirected graph we require that the edges be distinct. Graphs that have no cycles are called *acyclic*.

An undirected graph is *connected* if there is a path from every vertex to every other vertex. A directed graph with this property is called *strongly connected*. If a directed graph is not strongly connected, but the underlying undirected graph is connected, is called *weakly connected*.

A *complete* graph is a graph in which there is an edge between every pair of vertices.

3.1.2. Representation of Graphs

We are only interested in directed graphs, since we will be using them in this application. Suppose that we have the graph of Figure 4, which represents 7 vertices and 12 edges.

A simple way to represent a graph is to use a two-dimensional array. This is known as *adjacency matrix* representation. For each edge (u, v) we set A[u][v] to `true`; otherwise the entry in the array is `false`. If the edge has a weight associated with it, then we can set A[u][v] equal to the weight and use either a very large or a very small weight as a sentinel to indicate non-existent edges.

Although this has the merit of extreme simplicity, the space requirement is $T(|V|^2)$, which can be prohibitive if the graph does not have many edges. An adjacency matrix is an appropriate representation if the graph is dense:

$$|E| = \Theta\left(|V|^2\right)$$

In most of the applications that we may find, this is not true. If the graph is not dense, in other words, if the graph is sparse, a better solution is an adjacency list representation. For each vertex, we keep a list of all adjacent vertices. The space requirement is then

$$O(|E|+|V|)$$

which is linear in the size of the graph.

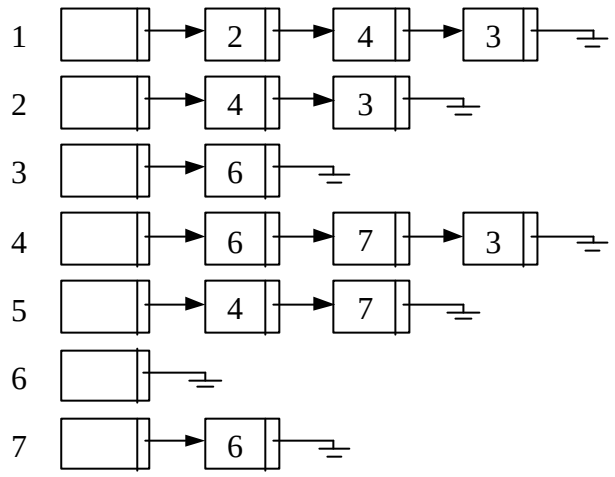

Figure 5 Adjacency List representation for the Graph in Figure 4

Adjacency lists are the standard way to represent graphs. A common requirement in graph algorithms is to find all vertices adjacent to some given vertex v, and this can be done, in time proportional to the number of such vertices found, by a simple scan down the appropriate adjacency list.

3.2. Applying Graph Theory to the Unit Selection Problem

To apply graph theory to our problem we regard all word instances as nodes of a graph. The edges of the graph then define the possible concatenations of the units. Because this graph looks very similar to a multi-layer perceptron network, we call all instances that belong to the same unit class a **layer**. It is easy to see that edges are only possible between subsequent layers and have a direction, which corresponds to

the order of time in the utterance. Each node in the first layer can be viewed as a possible start of the utterance. The same will happen in the last layer where each node is a possible end of the utterance. Because such a large number of start and end points are not practical we add two dummy nodes called **start** and **end node** to the graph. Then the start node is connected to each node of the first layer, and the each node of the last layer is connected to the end node no.

Supposing that we want to create the sentence "WordClass_1 WordClass_2", the graph shown in Figure 6 needs to be constructed.

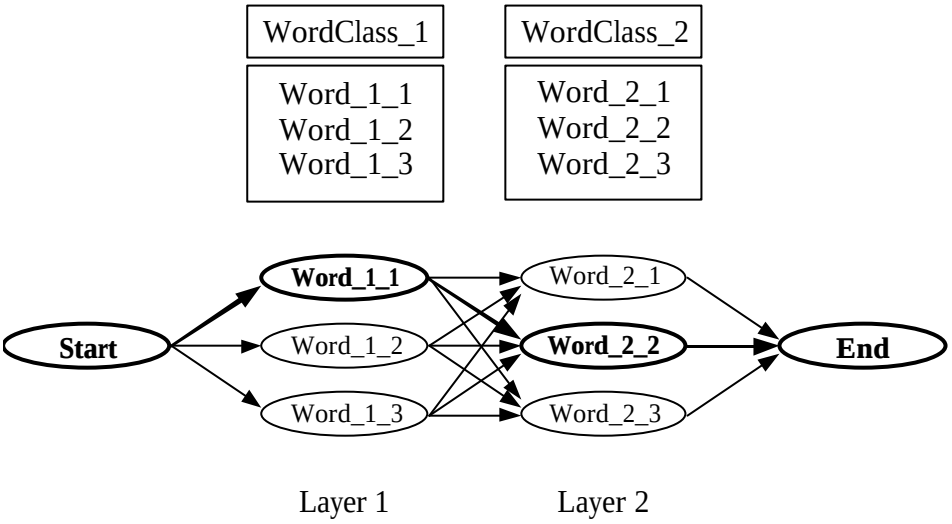

Figure 6 Graph of a Two-Word Sentence

Now we are able to define a **path** as a set of nodes connected by edges. The set of the highlighted nodes and edges in Figure 6 is a valid path.

Next we add a number to each edge in the graph. This number is the weighted sum given from a set of cost functions, and is called **edge cost**. The **cost of a path** is the sum of the values associated to the edges. Out aim is to find, among all paths containing the start and end node, the one that has the **minimal cost**. The path with minimal cost is called **shortest path**.

The reader may also want to check *Section 1.1. Background* in the Appendices, for a more formal definition of the problem.

3.2.1. Types of Costs

In our synthesis problem we have to distinguish between two types of costs. The first type, called **unit costs**, describes the usability of units without consideration of the unit instances in the neighboring layers. This might consist of values like the deviation between predicted and real duration of a unit instance. The second cost type, called **transition costs**, describes the transition between successive unit instances, like smoothness criteria for energy or F_0, or the consideration of the co-articulation between units in different layers.

There exist a lot of ways to apply the unit costs to the graph. In the Graph Creation file (see also *Section 3.3.2. Graph Creation Unit*), primarily for debugging reasons, we keep five cost fields for each edge in the input file used for the creation of the graph, each one representing one of the unit- and the transition costs described below. However, only the sum of these costs for each edge is finally stored in the Graph, as it is the sum that actually determines the weight of the edge.

3.3. Selection Criteria

Our knowledge about the construction of the synthetic utterance is associated with numerical values. For this reason we tend to use very simple functions to translate a property of a unit instance into a numerical value. A simple form of such a function is to do differentiation by cases: Assign cost 0, if the unit has the property, else assign cost 1. A set of those simple functions in conjunction with the determination of a shortest path forms a very complex rule system. It turns out that we need not understand all the complex dependencies implied by the cost functions. In most cases it is only necessary to add facts as new cost functions.

3.3.1. Cost Functions

On the word level the following cost functions are used:

- Concatenation Cost
- Coarticulation Cost
- Word Reduction Cost
- Word Position Cost
- Sentence Modality Cost

Concatenation Cost

If two units connected by an edge are not spoken consecutively in the corpus, 1 is added to the edge cost. Otherwise, no costs are assigned.

Coarticulation Cost

Modeling coarticulation for a given sequence of two words is done by comparing the last phoneme of the first word with the first phoneme of the second word.

We have to distinguish between the word sequence in the corpus and the word sequence we will synthesize. For each word in the corpus four phonemes (p, s, e, n) are additionally stored in our corpus description

p denotes the last phoneme of the previous word,
s the first phoneme of the considered word,
e the last phoneme of the considered word, and
n the first phoneme of next word.

For two unit classes u_1, u_2 connected by an edge, the expression

$$0.5 \cdot R_{eq}(u_1.e, u_2.p) + 0.5 \cdot R_{eq}(u_1.n, u_2.s)$$

bound by the interval (0, 1) is evaluated. The function $R_{eq}()$ defines a similarity relation for coarticulation between phonemes. The value of this expression is added to the edge.

The method that we use for the evaluation of the coarticulation cost function is based on the evaluation of the co-occurrence level of a given combination of phonemes.

Suppose that we want to synthesize the following sentence fragment:

"e??a? ?a???"

Suppose that we cannot find this fragment in the Weather Forecast Database. However, we do have the following two words recorded consecutively:

"e??a? ????a "

We also have "e??a?" recorded in sequence with other words. The problem that we are facing is which instance of "e??a?" to use to form the sentence fragment "e??a? ?a???".

We can make this decision by determining the level of similarity of the word "e??a?" when recorded before "?a???" and "????a ". We can the compare it to the similarity of the other instances of "e??a?", and use the one that matches, phonetically, best with "?a???".

The problem of the similarity of the edges of those previous two words can be reduced to the evaluation of the similarity of the following biphones:

e??**a??**a???	e??**a??**???a
[?]k	**[E]x**

Using the clustering information for the triphones observed in Greek, we may get an indication of the similarity of these biphones by examining the statistics for the following classes of triphones:

***[?]x-2** and ***[E]k-2**

where "-2" denotes the final state of the HMMs describing each class of triphones.

Thus, by computing the quantity:

$$c_{E-2}(x,k)=\sum_{\forall\ \text{clusteri}} \#(*[E]x-2)_i \cdot \#(*[E]k-2)_i$$

we can get an indication of the level of co-occurrence for the classes of triphones *[E]x and *[E]c. These are the triphones with [E] as the central phoneme, followed by x and c respectively, regardless of the phoneme preceding [E].

By computing the sum:

$$C_{E-2}(x) = \sum_{\substack{x, p_2 \\ x \neq p_2}} c_{E-2}(x, p_2)$$

we get the co-occurrences' sum of all the combinations between the *[E]x class of triphones and all the other triphones, that have E as their central phoneme.

Therefore, a metric for the co-occurrence of the two phonemes can be obtained by dividing these two quantities, yielding a value ranging from 0 to 1.

$$R(u_1.n, u_2.s) = R(x, k) = \frac{c_{E-2}(x, k)}{C_{E-2}(x)}$$

The equations used to calculate the coarticulation cost for any given pair of word instances, have the following general form:

$$Cost = 0.5R_{eq}(u_1.e, u_2.p) + 0.5R_{eq}(u_1.n, u_2.s)$$

where:

$$R_{eq}(u_1.e, u_2.p) = \begin{cases} 1 & \text{if } C_{u_2.s}(u_1.e) = 0 \\ 1 - \dfrac{c_{u_2.s}(u_1.e, u_2.p)}{C_{u_2.s}(u_1.e)} & \text{otherwise} \\ 0 & \text{if } u_1.e, u_2.p \end{cases}$$

with: $$c_{u_2.s}(u_1.e, u_2.p) = \sum_{\text{cluster i}} \#(*[u_2.s]u_1.e - 0)_i \cdot \#(*[u_2.s]u_2.p - 0)_i$$

$$C_{u_2.s}(u_1.e) = \sum_{x \neq u_1.e} c_{u_2.s}(u_1.e, x)$$

and

$$R_{eq}(u_1.n, u_2.s) = \begin{cases} 1 & \text{if } C_{u_1.e}(u_1.n) = 0 \\ 1 - \dfrac{c_{u_1.e}(u_1.n, u_2.s)}{C_{u_1.e}(u_1.n)} & \text{otherwise} \\ 0 & \text{if } u_1.n = u_2.s \end{cases}$$

with $$c_{u_1.e}(u_1.n, u_2.s) = \sum_{\text{cluster i}} \#(*[u_1.e]u_1.n - 2)_i \cdot \#(*[u_1.e]u_2.s - 2)_i$$

$$C_{u_1.e}(u_1.n) = \sum_{x \neq u_1.n} c_{u_1.e}(u_1.n, x)$$

Word Reduction Cost

If a word instance has the property *reduction* (see also *Section 2.3. Post-Recording Phase*) a cost of 1.9 is added to the interconnecting edge of the unit. Otherwise no cost is assigned.

In conjunction with the *Concatenation Cost* this will lead to the selection of a reduced word only if the left and right words are the left and right neighbors of the reduced word in the corpus.

Word Position Cost

The position of a word in an utterance may influence its prosodic structure. At least three different word positions have to be differentiated for spoken Greek. These are:

1. initial
2. final
3. neither 1) nor 2).

Normally we add 1 to the interconnecting edge of the unit if the requested word position is not equal to the denoted word position of the word instance. However, the quality of the synthetic speech decreases dramatically if a word instance with word position 2) is selected for a wrong position in the synthetic utterance. To avoid this case we add 3 to such an edge instead of 1.

Sentence Modality Cost

The sentence modality cost should distinguish between interrogative and declarative utterances. The F_0 curve is the most important perceptual cue for this distinction. A final fall of the F_0-curve will lead to a declarative intonation a final rise to an interrogative one. In our experience the F_0 curve of the last word is the primary indicator for the impression of sentence modality. For that reason each word in our corpus is labeled with a sentence modality attribute out of the set [i, d, u], where *i* denotes an interrogative, *d* denotes a declarative and *u* denotes an unknown F_0 curve. The synthesis input contains the sentence modality information so that a simple comparison between the requested and instance inherent modality will lead to the necessary cost function. Therefore, 0 is assigned to an interconnecting edge if the modalities match, otherwise 1.

The cost terms 1) and 2) belong to the **transition costs**, and 3) to 5) belong to the **unit costs**.

The following table summarizes the usage and the punitive costs assigned by the cost functions used for the unit selection problem.

Cost Function	Punitive Cost	Comment
Concatenation	1	Words not found consecutively in corpus
Coarticulation	$c \in [0, 1]$	Evaluated by expression $R_{eq}(u_1.e, u_2.p)+R_{eq}(u_1.n, u_2.s)$
Word Reduction	1.9	Word has the reduction property
Word Position	1 or 3	Applied to words found in different position.
Sentence Modality	1	Word modalities do not match.

Table 4 Cost Functions for the Unit Selection Problem

3.4. Shortest path algorithm

Selecting a path between two nodes of a weighted graph where the sum of weights assigned to the edges is minimal under all paths is a common problem in graph theory.

The input to a weighted shortest path algorithm is a weighted directed graph: associated with each edge (v_i, v_j) is a cost $c_{i,j}$ to traverse the edge. The cost of a path p = $\{v_1 v_2 \ldots v_N\}$ is

$$C(p) = \sum_{i=1}^{N-1} c_{i,i+1}$$

This is referred to as the **weighted path length**.

The **shortest path weight** from *u* to *v* is

$$d(u,v) = \begin{cases} \min\left\{w(p) : u \xrightarrow{p} v\right\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

The **shortest path** from vertex *u* to vertex *v* is then defined as any path *p* with weight

$$w(p) = d(u,v)$$

The identification of the shortest-path in the graph representing the possible concatenation options, given the initial vertex, is a classic **single source shortest-path problem**.

> ***Single Source Shortest-Path Problem***
>
> *Given as input a weighted graph, G = (V, E), and a distinguished vertex, s, find the shortest weighted path from s to every other vertex in G.*

By considering only the shortest paths initiating at Start node of the graph, the algorithm would provide us with the solution to the problem of finding the shortest weighted path from the Start node to every other vertex in the Graph, including the End node.

We have chosen to use Dijkstra's algorithm in order to solve the weighted shortest-path problem. This algorithm assumes that there are no negative edges. In our graph, all edges have costs equal to or greater than zero, so the algorithm can be used.

The running time for this algorithm is $O(|E|\log|V|)$ when implemented with reasonable data structures. More information of the idea behind the algorithm and its efficiency analysis can be found in *Section 1. Single-Source Shortest Paths* in the Appendices.

4. SIGNAL MANIPULATION

The average energy of the words in our corpus is considered during the recording process. But depending of the word context in the corpus there might be energy deviations at the concatenation points in the synthetic utterance. These deviations sound like plosives and disturb the natural sound of the synthetic utterance. To avoid this we do a simple energy smoothing operation on all words except the ones that are consecutively spoken in the corpus. Depending on the context just the left or right half of a 640-point Hamming window (thus applied to 20msec of speech signal) is multiplied with the samples near the left or right boundary of a word unit before concatenation is done.

Chapter 4

IMPLEMENTATION ISSUES

1. INTRODUCTION

In the previous chapter, we have presented the general guidelines of our approach to the problem of speech synthesis by word concatenation. However, we have neither analyzed most of the details of the procedure, nor explained our reasoning for most of our decisions at the technical level.

In this chapter we are going to provide insight to these technical aspects of our work, aiding the reader to understand our line of thinking that has lead us to this approach of the synthesis by concatenation problem.

2. CORPUS PREPARATION

2.1. Corpus Selection

The selection process of the sentences to form the corpus of the Weather Report Database, among those available to us in the material provided by EMY, can be described as follows:

> *Define a procedure for the selection of the most representative sentences to be used for the creation of the Corpus.*

The term *representative* is used to describe a sentence that, when added to the collection of sentences already chosen, introduces as much new information, in the form of words, either new or ones found in a context that has not yet been observed, as possible.

2.1.1. Corpus Selection Tools

In selecting these sentences, we were greatly assisted by the use of an implementation of a Concordance. A **concordance** of a file is a listing that contains all the words of that file, along with the line number on which the word occurs. So, for example, the following file:

Line No	
1	This is a line.
2	Another line follows the previous one.
3	One line.

Table 5 Sample File

has the following concordance:

a	1
Another	2
follows	2
is	1
line	1,3
line.	2
One	3
one.	2
previous	2
the	2
This	1

Table 6 Concordance of the sample file

We can use the concordance as an indication of the level of information of each sentence. Getting the concordance of the concordance of a given file does this. We will refer to the concordance of the original file as a *first level concordance*, and to the concordance of a first level concordance as a *second level concordance.*

First we alter the utility to be able to print the number of instances of a word in a file, rather than the lines where the word is found on. As we have already stated, the

concordance of the file contains all words in the file, along with the line numbers where they are found. Therefore, the second level concordance contains the line numbers of the original file and the line numbers where they are found in the concordance. By using the previous example, and by printing the counts rather than the line numbers, we get this listing:

```
1         4
2         6
3         2
a         1
Another   1
follows   1
is        1
line      1
line.     1
One       1
one.      1
previous  1
the       1
This      1
```

Table 7 Second Level Concordance of the Sample File

We just ignore the words – there is no use for them, since there is obviously only one occurrence of each word in the first level concordance – and concentrate on the first three lines. Each line contains two numbers. The first one is the number of a line from the original file. The second one is the count of the instances of that number (line) in the first level concordance, i.e. the number of words found in that line (sentence). Therefore, by inspection of the second level concordance we can get an indication of the line (sentence) that contains most new words.

2.1.2. Corpus Selection Algorithm

Suppose that we have a file with all the available sentences, and that we wish to select the most representative ones. Using the first and second level concordances, we can achieve that by following this procedure:

1. Find the line with the most occurrences.
2. Add this line to the Corpus and delete the words that it contains from the first level concordance.
3. Get the updated second level concordance, based on the new first level one, and repeat the procedure, until there are no more words left.

This procedure, simple though it is, works and produces adequate results. It can be seen as an entropy maximization procedure, since we always add to the corpus the sentence that will increase the entropy as much as possible.

However, this is a standard example of a greedy algorithm and several problems arise from the nature common to such a type of algorithms.

1. Each time, we are adding into the corpus the sentence with the most new words. This means that we will start by adding into the corpus the largest sentences, which usually include the most common words, leaving less common ones to be later included into the corpus. It's usually the case that after the first 5 – 10 sentences, where at least 80% of their words are new to the corpus, we will be adding 10 – 15 word sentences, in order to add 3 – 5 new words to the corpus.
2. We treat common and rare words in the same way.
   We should have given rare words a greater weight factor, leading to their selection in the beginning, rather than at the end of the procedure. By doing that, we would also be adding more common words along with the rare ones, eliminating the need to add them *per se* in another sentence.

These seem to be severe problems of the procedure, but only if we see the procedure without bearing in mind the actual task that we are performing. Although the corpus produced in this manner is not the smallest possible, we actually want this to happen. We need words with relative higher occurrence rate in the original data set to have a

high occurrence rate in the corpus too. We have observed that even though sub-optimal, with concern to the number of the corpus's words, this selection procedure preserves the statistical characteristics of the original data set.

3. GENERAL OVERVIEW

The final application is actually a linear application of several procedures to different data, leading to the desired result. This proved to be very useful since we could develop each stage of the procedure independently of the others, knowing only the type and form of messages that were to be passed down from one stage to the next. It also allowed us to tweak the performance of each stage, improving the overall quality of the synthesized speech, without the need for extensive changes in the architecture of the application.

There are also some procedures that had to be applied only once on certain sets of data, creating new data that are needed by the final application. These data were needed by all stages of the application and had to be created before the synthesizer could work. Data such as the Word Segment Database and the Coarticulation Database belong to this category.

This section will be providing insight into the works of our synthesizer, explaining each stage of the synthesis procedure.

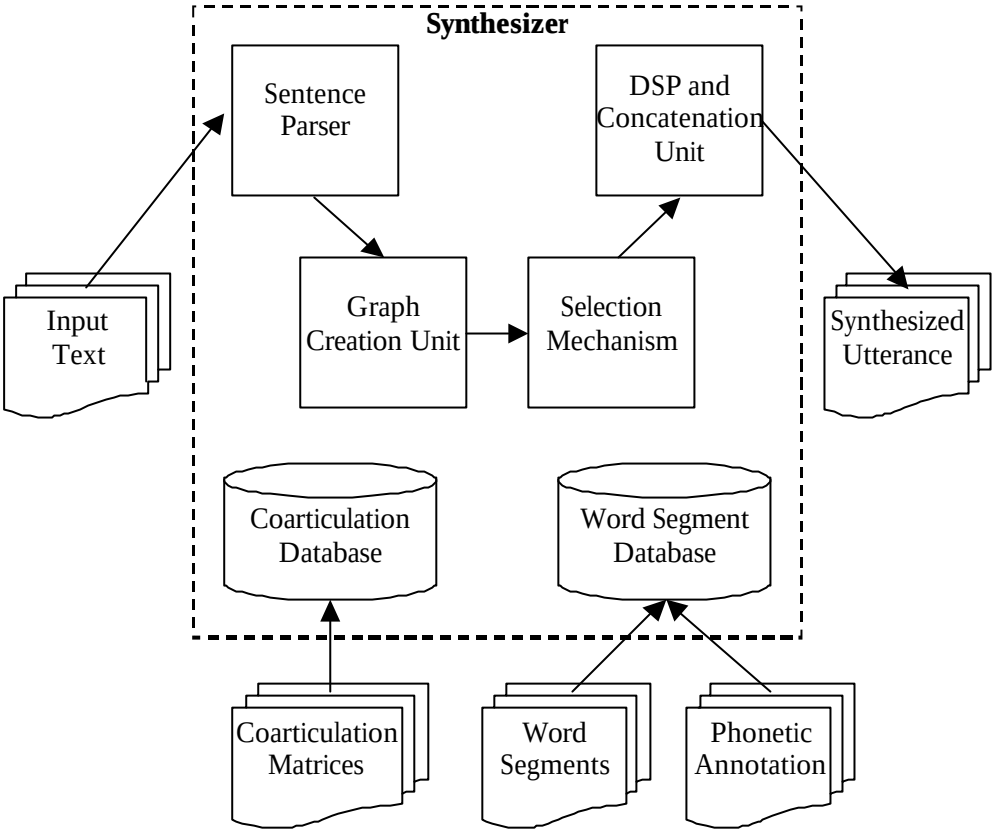

Figure 7 Components of Speech Synthesizer

3.1. Internal Data

The basic function of our Text-to-Speech synthesizer is to transform a given sequence of words into their associated spoken representation. Thus, the input file, containing that very sequence of words, forms the starting point of the speech creation process.

However, the system cannot function without the presence of the Coarticulation and the Word Segment Databases. These parts of the system need to be present before the input sentence file can be parsed and processed.

3.1.1. Word Segment Database

The Word Segment Database forms an easy to use and quick in response medium that serves both as storage of the recorded instances of the words found in the corpus, and as a retrieval mechanism for those instances.

We designed the database to support the following two services:

store a specific recorded instance and associate it with the word it represents
retrieve a specific recorded instance

The performance of our synthesizer relies heavily on access to these recorded instances that are added to the database during the start-up period of our application. Thus both the "store" and "retrieve" operations needed to take as little time as possible.

Words

Each entry to the database represents a certain recorded instance of a word and contains all information associated with that instance. Table 8 summarizes the data stored along with each word instance.

Field	Use
wordName	Name of Recorded Instance
uttName	Name of Utterance containing that Word Instance
uttPosition	Position of Word Instance in Utterance
maxPosition	Number of Words in Utterance
p	Last phoneme of previous Word in Utterance
s	First phoneme of Word
e	Last phoneme of Word
n	First phoneme of next Word in Utterance
reduced	Word has the reduction property
position	Relative position (initial, final, medial) in Utterance

Table 8 Word Instance Data

The **wordName** is the key attribute and identifies each Recorded Word Instance uniquely. The **uttName** and the **uttPosition** attributes are used during the selection process to determine their *Concatenation Cost.* If the two instances share the same **uttName** and have consecutive **uttPosition**s then they can be concatenated with no cost. The **uttPosition** and the **maxPosition** attributes are used to determine the **position** attribute, labeling the instance as found at the start of the utterance, at its end, or neither at the start or at the end. The **reduced** attribute has the use discussed in *Section 2.3. Post-Recording Phase* in Chapter 3. The attributes **p, s, e, n** are used as stated in the description of *Coarticulation Cost* in *Section 3.3.1. Cost Functions* also in Chapter 3.

One notices that there is no reference to the Word Class to which the Recorded Word Instance belongs. The reference exists but is not immediate, since we have given a hierarchical structure to the corpus database. This can be visualized in Figure 8.

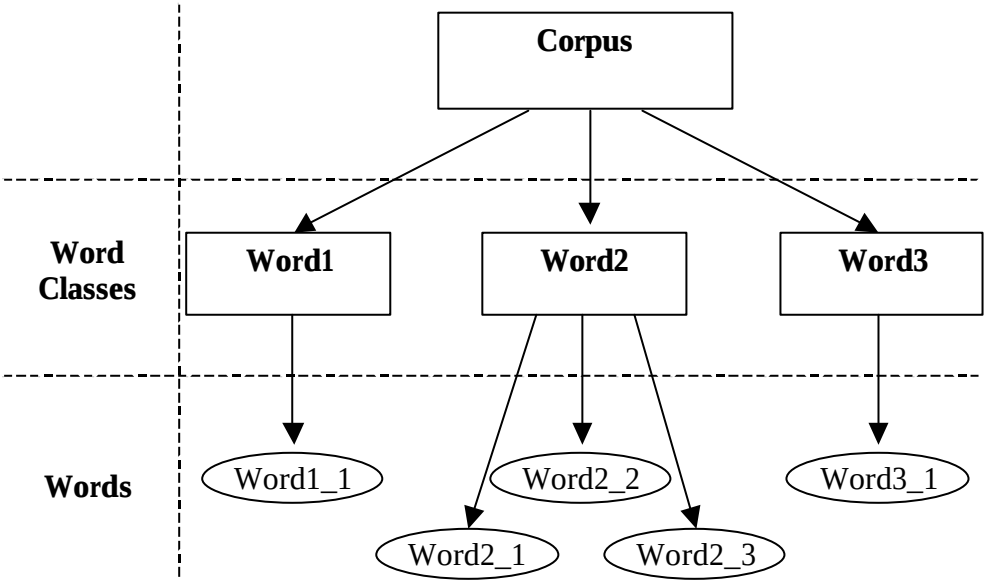

Figure 8 The Hierarchical Structure of the Recorded Segments Database

WordClasses

Each recorded word instance is inserted into the database under the Word Class whose orthographic form is the same as that of the current recorded instance. If no such Word Class exists, a new one is created and the Word instance is inserted under that.

In each **WordClass** object, the information shown in Table 9 is kept:

Field	Use
className	Name of Word Class
wordMap	Mapping structure for all Word Instances of Class
allWords	List with Word Instances belonging to Class
numWords	Number of Word Instanced belonging to Class

Table 9 Word Class Data

The **className** is a unique identifier of a WordClass object and serves as the key attribute. The **numWords** attribute is used to get the number of word instances belonging to a certain word class, without needing to access the **allWords** list, where they are stored. Finally, **wordMap** provides a quick access to the recorded word instances of this Word Class, using the **wordName** attribute of each Word as the key.

Corpus

Finally, all **WordClasses** belong to a single **Corpus**. The function of the Corpus object for WordClasses is similar to the function of WordClasses for Words. It encapsulates all the WordClasses and provides the means to access them quickly by using their className as the key.

A **Corpus** object contains the information shown in Table 10:

Field	Use
wordClassMap	Mapping structure for all Word Classes in Corpus
allWordClasses	List with Word Classes belonging to Corpus
numWordClasses	Number of Word Classes belonging to Corpus

Table 10 Corpus Data

Maps

The maps used to provide access to the WordClasses of the Corpus and the Words of each WordClass actually consist of a collection of pairs of attributes tied together in a single construct. For instance, the **wordMap** of any WordClass is a collection of the following pairs:

Field	Use
wordName	Name of Word Instance
Word*	Pointer to that Word

Table 11 wordMap Data

This map allows the retrieval of the Word associated with a given wordName from the pool of Word instances of that WordClass. However, this implies that there exists a method that allows us to find that specific wordName among the other wordNames as quickly as possible. This is achieved by implementing the map as a **Hash Table**.

Hashing

There are several reasons for choosing the map to be implemented as a Hash Table. The most important is the fact that hashing allows insertions, deletions and finds to be performed in constant average time. We have discussed the speed factor above and presented our reasoning on why the retrieval and storage operations for the Word Segment Database need to be as quick as possible. Hashing, as implemented through *Separate Chaining* is consistent with the requirements presented above.

The efficiency of hashing is based both on the data structure used to implement the hash table and on the hashing function used to determine the place where each element is to be stored in this table. Collision resolution, i.e. the strategy to be followed when two keys hash to the same value, and the hash table size also play a significant role on the speed of a certain hash table implementation.

We chose the separate chaining method, since it provides an easy way to resolve collisions, by keeping a list of all elements that hash to the same value, while at the

same time the correct selection of the hash table's size allows for a constant average time for the insertion and retrieval operations on the hash table. This can be achieved by setting the size of the table at least equal to the average number of elements that are to be stored in the hash table. Thus, the load factor ? of the hash table, that is the ratio of the number of elements in the table to the table size, would be equal to ? = 1.0. Since the effort to perform a search is the constant time required to evaluate the hash function, plus the time to traverse the list. The average length of each list is ?. In an unsuccessful search, the number of nodes to examine is ? on the average. A successful search requires that about 1+(?/2) links are traversed.[2]

The hash function that we have used involves all the characters in the key (the name of the Word or WordClass) and can generally be expected to distribute well. It computes

$$hash = \sum_{i=0}^{KeySize-1} Key[KeySize - i - 1] \cdot 37^i$$

and brings the result into proper form. It uses Horner's rule to compute a polynomial function. It also takes advantage of the fact that overflow is allowed, and that it can introduce a negative number, hence there is an extra test. Given the length of the keys that are used in our application, the hash function is both simple and reasonably fast.

Segment Naming Policy

We have seen that the name file containing the fragment that is inserted into the Word Segment Database is used as the key for both its insertion and retrieval. We have chosen to use the following convention regarding the naming of these files.

```
uttxxx_yyy_zzz.wav
```

where:

[2] To see this, notice that the list that is being searched contains the one node that stores the match, plus zero or more other nodes. The expected number of "other nodes" in a table of N elements and M lists is (N-1)/M = ? – 1/M, which is essentially ?, since M is presumed large. On average, half the "other nodes" are searched, so combined with the matching node, we obtain an average search of 1 + ?/2 nodes.

xxx: the number of the corpus utterance from where the fragment was extracted
yyy: the position of the word instance in the utterance
zzz: the total number of words in the utterance

Figure 9 demonstrates our naming scheme:

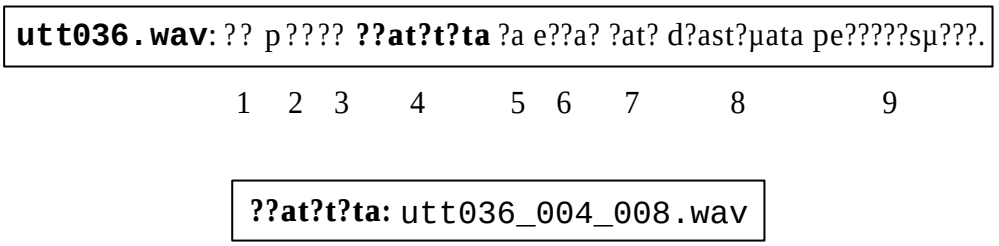

Figure 9 Segment Naming Example

3.1.2. Coarticulation Database

The inclusion of the Coarticulation Cost as a criterion for the concatenation of any given pair of Words was found to have notable results on the quality of the synthesized speech. This is attributed to the nature of the data used to support the cost assignment for this case.

The Coarticulation Cost for a given pair of Words is the result of a two level similarity function. Costs due to the use of both the first and the second Word are added and normalized to a maximum value of 1.0. We have already discussed the way this cost function works in *Section 3.3.1. Cost Functions* in Chapter 3. In this section we will be discussing the way these costs are assigned.

Our intention is to determine the similarity level of any two instances of a certain phoneme when followed or preceded by some other phoneme. For instance, we want to determine how much similar is the phoneme [E] in these two instances:

"e??**a?**?a???" and "e??**a?**????a"

This can be achieved by using the clustering information for all triphones as found in the Nuance® speech recognition system. Triphones are segments of speech

consisting of three phonemes. They allow for the representation of the central phoneme in the context of its neighboring phonemes. For instance, the "a?" in "e??a? ?a???" has the following triphone representation:

"e??**a?**?a???" ? n[E]c

Triphones and HMMs

In an HMM-based speech recognition system triphones are represented as three-state HMM processes. Transitions from each state to the other model the temporal change of the speech signal for that particular triphone. Each state has an output distribution that is associated with the acoustical vector of that particular segment of the triphone. Thus, each state represents part of the spectral features of the triphone.

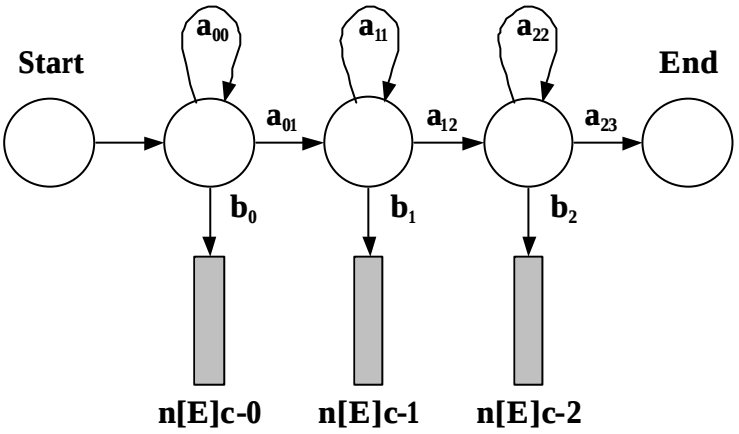

Figure 10 HMM for triphone n[E]c

Figure 10 shows the HMM representation of the triphone n[E]c. Each state produces as output a spectral feature vector. Due to the relatively large number of triphones in Greek (as well as in other languages) – approximately 14,000 – a clustering scheme is used to reduce the number of distinct spectral feature vectors.

Clustering

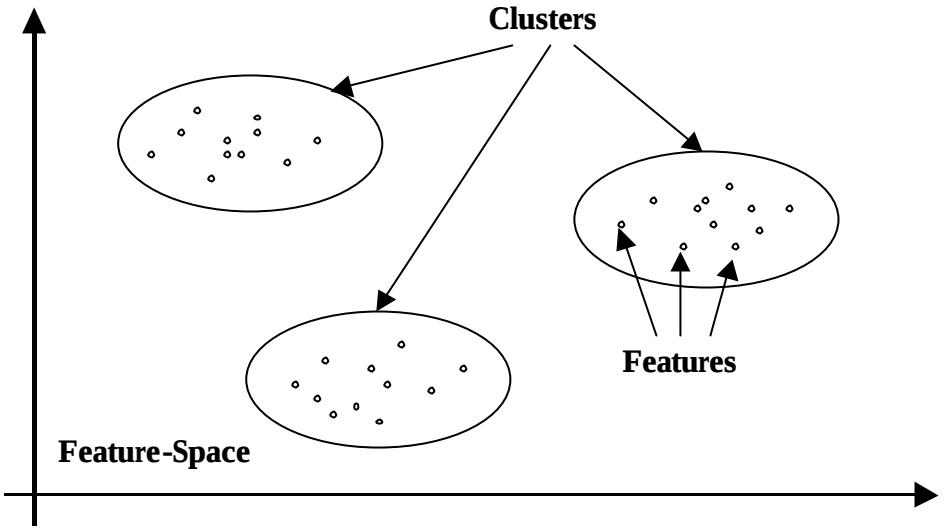

Figure 11 Clustering of Feature Vectors

According to this scheme, feature vectors are clustered together according to their elements' similarity. Thus, a single vector, representing the whole collection, replaces each collection of similar feature vectors. Figure 11 allows us to visualize the function of the clustering process.

The number of clusters that are going to be formed varies according to the implementation of the clustering scheme that is used. We have made use of the clustering information for triphones for the Greek language, as used in the Nuance® speech recognition system, and in that case the 14,544 triphones were merged into 986 clusters.

```
#[z]o-1     970
i[z]o-1     970
u[z]A-1     970
o[z]#-1     970
o[z]E-1     970
o[z]A-1     970
E[z]o-1     970
A[z]o-1     970
E[z]o-2     971
#[z]o-2     971
t[z]o-2     971
i[z]o-2     971
A[z]o-2     971
E[z]i-0     972
```

Figure 12 The Clustering Information File

Since similarity of feature vectors is used as a criterion for the clustering procedure, we may use the Clusters Info File – an excerpt is shown in Figure 12 – to extract the information that may be used to obtain a notion on the phonetic similarity of any given set of triphones.

This can be understood by examining the previous example that we have used in the *Coarticulation Cost* section of Chapter 3. Suppose we want to form the phrase:

"e??a? ?a???"

when the following speech excerpts exist in the Corpus, among others containing either of the two Word Classes that we want to concatenate:

"e??a? ????a" and "p??? ?a???".

The former phrase may be constructed through the concatenation of the words "e??a?" from the first excerpt, and "?a???" from the second one. However, the two words are recorded in different context than the desired. This can be seen in Table 12. "*" stands for any phoneme.

Word	Desired	Available
e??a?	*[E]k	*[E]x
?a???	E[k]*	i[k]*

Table 12 Words in Context

We need a measure of how similar the word instance "e??a?" is when recorded before a word starting with [k] and another one starting with [x]. We also wish to have this information for the word "?a???" when recorded after a word ending in [E] and another one ending in [i].

Remember that a triphone is represented by a three-state HMM. The first state of the HMM represents the beginning of the triphone, while its last state represents its end. Thus, when judging the similarity of the beginning of a triphone with another triphone, only the feature vectors of their first states need to be used to extract any information. In the case when we want to determine the similarity of the end of two triphones, the last states' feature vectors need to be consulted.

It is obvious that the decision is based on two different kinds of information, depending on whether the beginning or the end of the word is being considered.

Coarticulation Matrices

This information is found in what we call the two *Coarticulation Matrices*, one for each situation mentioned above. These matrices are constructed directly from the Clustering Information file. We will be presenting how the matrix used for the evaluation of the phonetic similarity of the first word in a pair is constructed.

Each cluster contains feature vectors of several triphones. These feature vectors are phonetically equivalent, meaning that their respective triphones have similar pronunciations. When considering the first word in a pair, we are only interested in its end. Thus, we are interested only in the last feature vectors of the respective triphones. Those vectors are marked with a "-2" in the clustering info file. In addition

to that, we do not care for the first phoneme of the triphone. Thus the vectors "a[E]k-2" and "b[E]k-2" are regarded as equivalent in this context.

We will call the collection of feature vectors such as ***[y]z-n** and **x[y]*-n** as ***feature vector classes***. As we have already mentioned, the phonetic similarity of the classes of feature vectors *[E]x-2 and *[E]k-2 can be evaluated by using the following equation

$$R(u_1.n, u_2.s) = R(x,k) = \frac{c_{E-2}(x,k)}{C_{E-2}(x)} \quad (1)$$

where

$$c_{E-2}(x,k) = \sum_{\forall \text{ clusteri}} \#(*[E]x-2)_i \cdot \#(*[E]k-2)_i \quad (2)$$

$$C_{E-2}(x) = \sum_{\substack{x,p_2 \\ x \neq p_2}} c_{E-2}(x,p_2) \quad (3)$$

Equation (1) produces an indication of how probable is the substitution of the triphone *[E]k by the triphone *[E]x. It is based in the level of co-occurrence of the feature vector classes *[E]k and *[E]x in any given cluster. Equation (2) is a measure of the co-occurrence since it is the sum of the product of the occurrences of the two vectors classes in the same cluster. Finally, equation (3) provides a measure of the co-occurrence of the vector class *[E]x-2 with any vector class with [E] as the central phoneme in all clusters.

The Coarticulation Matrices provide us with the information needed to evaluate equation (1), thus facilitating the evaluation of the Coarticulation Cost for any given combination of phonemes. They are loaded at the beginning of the application, thus creating the Coarticulation Database. These matrices organize the information found in the Clustering Information file, and allow us to easily gain access to this information. For more information on the Coarticulation Matrices, see *Section 3.2.3. Coarticulation Matrices*.

3.2. Input Data

The basic input data for any Text-to-Speech system is the sequence of characters forming the sentence to be synthesized. However, we will also be considering the data used for the construction of the internal structures described in *Section 3.1. Internal Data*.

3.2.1. Sentence File

The file containing the sentence to be synthesized is simple in its syntax. It contains the sentence, along with any punctuation marks that are to be applied to the speech, enclosed in the symbols "$s" and "$f" at the beginning and the end of the sentence.

```
$s <word1> <word2> ... <wordn> $f
```

Figure 13 The syntax of the Sentence File

The use of those two symbols is going to be made clear later, however we may say that they represent the "Start" and "End" nodes of the graph that will be created later on for this sentence.

3.2.2. Word Metadata File

This file is used during the creation of the Word Segment Database. It contains the metadata associated to each Recorded Word Instance extracted from the sentences of the Corpus.

The metadata consists of the names of the WordClasses and their associated Words, and for each Word of the phonemes at the edges of the Word and the last and first phoneme of the previous and next word, respectively, in the corpus. Finally the reduced attribute for each Word is also included.

The information in this file has the syntax shown in Figure 14.

```
<word1>
<word_instance1>
<p> <s> <e> <n> <reduced>
<word_instance2>
<p> <s> <e> <n> <reduced>
...
<word_instanceN>
<p> <s> <e> <n> <reduced>
%
<word2>
<word_instance1>
<p> <s> <e> <n> <reduced>
...
%
<wordN>
<word_instance1>
<p> <s> <e> <n> <reduced>
...
%
```

Figure 14 Syntax of the Word Metadata File

All information found in the Metadata file, except for the reduction characterization, is inserted automatically using the Segment Alignments file (see *Section 2.3. Post-Recording Phase* in Chapter 3). However, the reduction property is evaluated by listening to each Word Instance in order to determine whether it may be characterized as "reduced".

3.2.3. Coarticulation Matrices

These files are used for the construction of the Coarticulation Database. They are the dumps of the matrices. Therefore, the matrices can be easily reconstructed just by reading these files.

The syntax of the files is simple. As shown in Figure 16 and Figure 18, they begin with the dimensions of the matrix to be constructed and are followed by the matrix's elements. The files are in ASCII format for easy inspection.

As stated before, the coarticulation cost is evaluated as the result of the following equation:

$$R(u_1.n, u_2.s) = R(x, c) = \frac{c_{E-2}(x, c)}{C_{E-2}(x)}$$

Both factors of the fraction that results to the coarticulation cost are stored in those matrices rather than computed at run-time. The first matrix – shown in Figure 15 – may be called the Co-Occurrence matrix, since it provides a measure of the co-occurrence of two phonemes, before or after another phoneme.

$$c_{center-2}(opt2, opt1) = \sum_{\forall \text{ cluster } i} \#(*[center]opt2-2)_i \cdot \#(*[center]opt1-2)_i$$

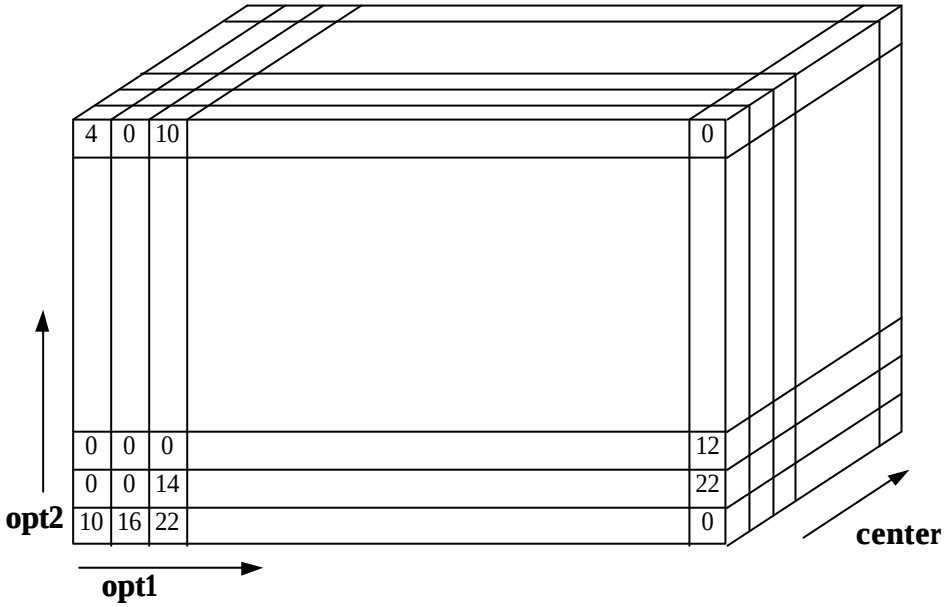

Figure 15 The Co-Occurrence Matrix

In each cell of the matrix, a value indicating the number of co-occurrences of the triphone classes, consisting of the central phoneme and the two phonemes in question, in the same cluster is found. Thus, the evaluation of the *c()* function is reduced to a simple lookup to the Co-Occurrence matrix, using the three phonemes as indices.

```
<opt1_num> <opt2_num> <center_num>

<elm_1-1-1> <elm_1-2-1> ... <elm_1-N-1>
<elm_2-1-1> <elm_2-2-1> ... <elm_2-N-1>
...
<elm_N-1-1> <elm_N-2-1> ... <elm_N-N-1>

<elm_1-1-2> <elm_1-2-2> ... <elm_1-N-2>
<elm_2-1-2> <elm_2-2-2> ... <elm_2-N-2>
...
<elm_N-1-2> <elm_N-2-2> ... <elm_N-N-2>

...

<elm_1-1-N> <elm_1-2-N> ... <elm_1-N-N>
<elm_2-1-N> <elm_2-2-N> ... <elm_2-N-N>
...
<elm_N-1-N> <elm_N-2-N> ... <elm_N-N-N>
```

Figure 16 The syntax of the Coarticulation Matrices

The denominator of the R() function, representing the total co-occurrences of a phoneme with all other phonemes, along with the central phoneme, is also reduced to a simple lookup of the Total Co-Occurrence Matrix, shown in Figure 17.

$$C_{center-2}(opt2) = \sum_{\substack{x, p_2 \\ x \neq p_2}} c_{center-2}(opt2, p_2)$$

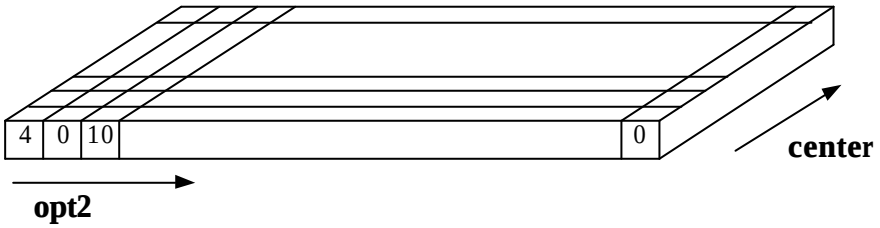

Figure 17 The Total Co-Occurrence Matrix

It is obvious that this is a two-dimensional matrix, since there are only two features of concern, the central phoneme and the phoneme whose total co-occurrence level is of question. However, for uniformity, we use the same syntax for the data in this matrix, regarding it as a three-dimensional matrix, with one dimension equal to 1.

```
<opt2_num> 1 <center_num>
<elm_1-1-1> <elm_1-2-1> ... <elm_1-N-1>
<elm_1-1-2> <elm_1-2-2> ... <elm_1-N-2>
...
<elm_1-1-N> <elm_1-2-N> ... <elm_1-N-N>
```

Figure 18 The syntax of the Total Co-Occurrence Matrix

3.3. The Synthesis Process

We have already described the data structures needed to support the synthesizer. We are ready to study the process initiated by reading the Sentence File and leading to the synthesis of the associated speech in the form of a Waveform File.

3.3.1. Sentence Parser

The Sentence Parser reads the input sentence from the Sentence file and creates a List with the words that form this sentence. These words are actually the WordClasses that will be used for the creation of the Graph later on.

$s <Word1> <Word2> … <WordN> $f

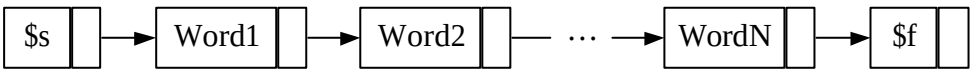

Figure 19 Creation of Word Class List from Sentence File

3.3.2. Graph Creation Unit

Using the list of WordClasses created by the Sentence Parser, the actual Word Graph is created.

There are two basic functions for the Graph Creation unit. The first is to make sure that this sentence can be synthesized. It is obvious that we cannot synthesize a

sentence if there is not at least one available Word Instance for every Word Class in the sentence. Its second function is the actual creation of the Graph.

The first function is easily accomplished just by checking if each of the Word Classes in the List has a non-empty *allWords* list. Remember that a WordClass's *allWords* list holds all the Word instances of that WordClass.

The second function is actually just simulated at this step. The Graph is created later, by a procedure using as input a Graph Creation File created now. This file contains pairs of vertices, indicating the edges of the graph, and the costs associated with these vertices and edges. In a further stage, this file is used to create the graph as a collection of interconnected vertices, with a vertex ($s) serving as the start, and another ($f) as the end of the graph, incorporating information on the transition and unit costs into the vertices.

Representation of Graphs

In the case of our application we are working with sparse graphs. All vertices are connected only with the vertices of the next layer. Thus, instead of using *adjacency matrices*, a better solution is the *adjacency list* representation. This means that, for each vertex, we keep a list of all adjacent vertices. The space requirement is then

$$O(|E| + |V|)$$

which is linear in the size of the graph.

In the case of weighted graphs, the weight of the edge is also included in the Adjacency List representing the graph.

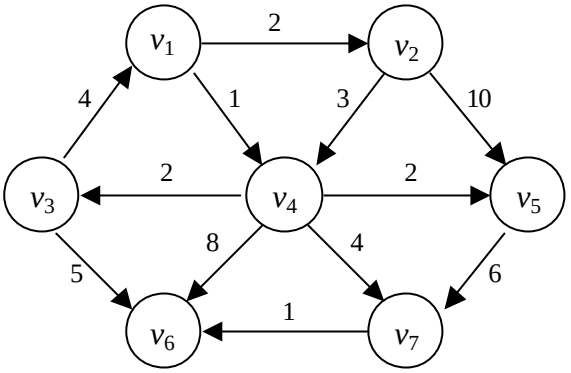

Figure 20 A directed, weighted graph

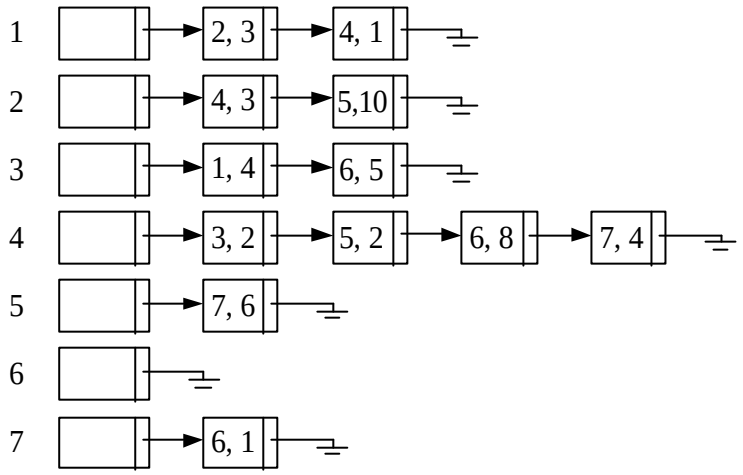

Figure 21 Adjacency List representation for the Graph in Figure 20

Adjacency lists are the standard way to represent graphs. A common requirement in graph algorithms is to find all vertices adjacent to some given vertex *v*, and this can be done, in time proportional to the number of such vertices found, by a simple scan down the appropriate adjacency list.

Information about each vertex, including a list of its adjacent vertices, is stored in an object of type Vertex. In most real-life applications, the vertices have names, which are unknown at compile time; and thus, generally, we will need to provide a mapping of the names to its corresponding Vertex object. The easiest way to do this is to use a hash table, in which we store a name (which serves as the key) and a pointer to a

Vertex. New Vertex objects are created as the graph is read. As each input, we check whether each of the two vertices has already been seen. If so, we use the Vertex corresponding to it. Otherwise, we create a new Vertex object and insert the name and Vertex object as a pair into the hash table. Each Vertex entry will also need to store the vertex name, since, eventually, we will need to output these names.

Creating the Graph File

The Graph File that we have mentioned above has a syntax shown in Figure 22. The '#' symbols act as inline comments for the aid of the user. The two Words are vertices of the graph and define an edge. The sum of the unit cost associated with the first word and the transition cost associated with that pair of words is assigned to the edge defined by these two Words.

```
#<WordClass1>
<Word1_1> <Word2_1> <Unit_Cost1_1> <Trans_Cost1_1-2_1>
<Word1_1> <Word2_2> <Unit_Cost1_1> <Trans_Cost1_1-2_2>
...
<Word1_1> <Word2_N> <Unit_Cost1_1> <Trans_Cost1_1-2_N>

#<WordClass2>
<Word2_1> <Word3_1> <Unit_Cost2_1> <Trans_Cost2_1-3_1>
...
<Word2_M> <Word3_K> <Unit_Cost2_M> <Trans_Cost2_M-3_K>

...

#<WordClassN>
```

Figure 22 The syntax of the Graph Creation File

The creation of the graph file is simple in its conception. Using the WordClasses present in the WordList created by the Sentence Parser, we traverse the list considering every time the current WordClass, and the following one. Using the allWords list associated with each WordClass, we form all the pairs of Word Instances of those two WordClasses.

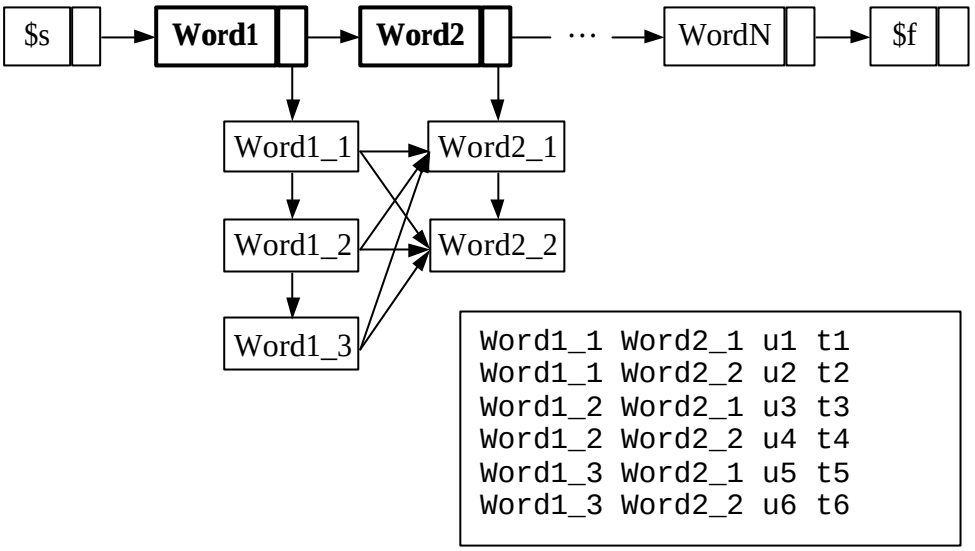

Figure 23 Creation of Graph from Word Class List and Graph Creation File

The unit cost of each Word Instance is the sum of these factors:

Unit Cost = Word Reduction Cost + Word Position Cost

The Word Reduction cost is determined by the value of the *reduced* field in the Word structure. The *position* field in the same structure determines the position of this Word Instance in the sentence it was extracted from. This is compared to the actual position of this WordClass in the sentence, and the Word Position cost is computed.

The transition cost of any given pair of Word Instances is the sum:

Transaction Cost = Concatenation Cost + Coarticulation Cost

The Concatenation cost is determined by comparing the *UttName* and *UttPosition* attributes of each Word Instance. If both instances have the same *UttName* and consecutive *UttPosition* values, then they were recorded consecutively. Otherwise, a certain penalty should be applied for their concatenation. The phonemes stored with each Word Instance are used as input to the Phonetic Similarity function, producing an estimation of the Coarticulation Cost.

Creating the Graph

As we have already stated, the Word Graph is formed by reading the Graph Creation File. The Word Graph has been implemented as a collection of Vertices connected, through Edges of certain cost, to other Vertices.

Each Vertex caries the information shown in Table 13.

Field	Use
name	Name of Vertex
adj	List of Edges with adjacent Vertices in Graph
known	Set when Vertex has been selected
dist	Cost of shortest path so far
path	Previous Vertex on Shortest Path

Table 13 Vertex Data

The **name** of each Vertex is actually the name of the Word Instance being represented by the Vertex. However, since the possibility exists that a certain Word may be found more than once in the same Sentence, meaning that two different layers in the Graph may consist of the same Words, an additional number is augmented at the end of the name, indicating the layer in which this Vertex is located. Thus all Vertices are unique.

The **known**, **dist** and **path** fields and are necessary for the function of the Shortest Path Algorithm. Finally, the **adj** list holds a list of the Edges starting from the current Vertex. This list actually implies the form of the Graph, and is created by reading the Graph Creation File. All Edges that begin with this Vertex are included in this list. The list also implements the directionality of the Graph. This can be seen in Figure 24 where only the Edges originating from the current Vertex are included in the **adj** list.

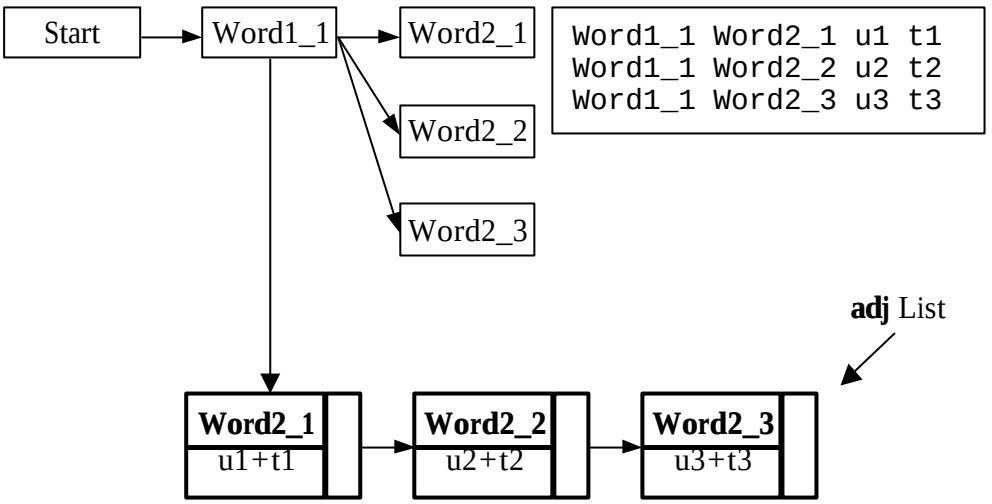

Figure 24 Use of **adj** List in Graph

Edges enclose the information shown in Table 14.

Field	Use
pVertex	Vertex at end of Edge
cvw	Edge Cost

Table 14 Edge Data

The **pVertex** attribute holds a pointer to the actual Vertex at the end of the Edge, and **cvw** the cost assigned to that Edge.

The data used to implement the Graph is shown in Table 15.

Field	Use
vertexMap	Mapping structure for all Vertices of Graph
allVertices	List with Vertices belonging to Graph
numVertices	Number of Vertices belonging to Graph

Table 15 Graph Data

Like the structures used for the Word Segment Database, the Vertices forming the Graph are stored in the **allVertices** List, with the **vertexMap** mapping structure

providing quick access to any Vertex, using its name as the key. The **numVertices** attribute stores the number of individual Vertices in the Graph.

3.3.3. Selection Mechanism

The Selection Mechanism is implemented by applying Dijkstra's algorithm on the Graph. This algorithm finds the shortest path between two Vertices. In our application, we are only interested in the shortest path between the "Start" and "End" Vertices of the Graph.

The path can be found by tracking back to the origin ("Start") Vertex starting at the destination ("End") Vertex. This procedure produces the series of Word Instances that should be used for the synthesis of the input sentence.

The DSP and Concatenation Unit however synthesizes the sentence by using this series of Word Instances as input. In fact, the synthesis is based on the Fragments File, the file containing the exact locations of each one of the waveform files containing these Word Instances.

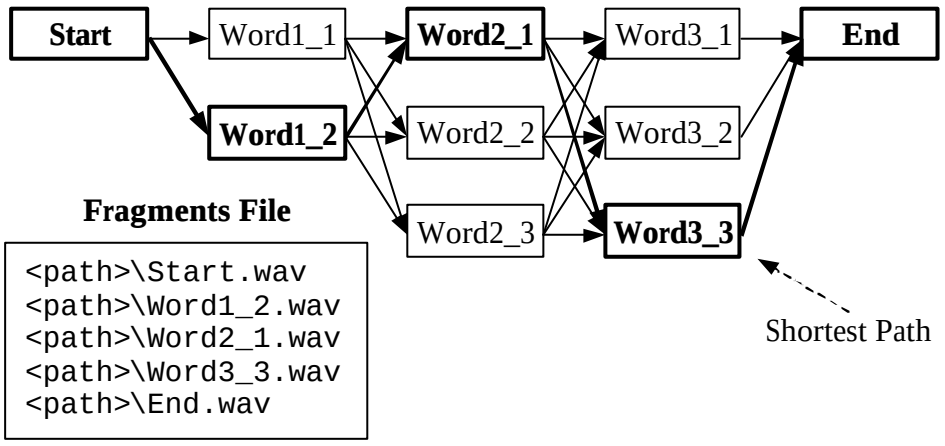

Figure 25 Creation of the Fragments File

3.3.4. DSP and Concatenation Unit

The Selection process provides as input to the DSP and Concatenation Unit the Fragments File, containing the exact path and filename of each Segment to be used for the creation of the synthesized sentence.

Digital Signal Processing

According to the strategy that we have followed during the segment selection phase, the use of consecutive word segments, extracted from the same utterance of the corpus, is promoted over the use of isolated segments. However, only sometimes is this the case, so the concatenation of isolated words is necessary for the synthesis of the desired sentence.

The energy levels at the edges of most words are usually different. For instance, many word segments start or end at a high energy level, and it has been observed that most segments have a great difference at their energy levels at the edges. This energy level mismatch has the effect of introducing annoying noises in the synthesized speech, reducing overall quality, thus making the speech more difficult to listen to.

We have chosen to attack this problem by introducing a simple energy smoothing operation on the edges of segments that are to be concatenated with other, non-consecutive, segments. We achieve that by applying the left or right half of a 640-point Hamming window to the edges of the segments that are to be concatenated. This means that the last 20 msec of the left segment are faded out and the first 20 msec of the right segment are faded in before being concatenated into the synthesized sentence.

Concatenation

The Segments that are used for the synthesis are simple waveform files. The concatenation of the waveform files produces the synthesized sentence. These waveform files are stored in RIFF format (See Also *Section 2. Resource Interface Format Files* in the Appendices). The synthesized sentence is also stored in RIFF format.

The concatenation procedure consists of the creation of a new RIFF waveform file, whose data subchunk is the collection of the data subchunks of the individual files that form the synthesized sentence.

Chapter 5

QUALITY EVALUATION

1. INTRODUCTION

In spite of the rapid progress that is being made in the field of speech technology, any speech synthesis system available today can still be spotted for what it is: nonhuman, a machine. Although there have been significant improvements in the quality of the output of TTS systems, as long as synthetic speech is inferior to human speech, synthesis evaluation will be useful.

Speech synthesis assessment can be important to two parties: systems designers on the one hand, and prospective buyers and end user on the other. Designers are intent on improving their TTS-systems. However, designers who have grown up with their systems are used to all its habits; they are likely to understand its output better than first-time users, and will often overrate its performance level. More meaningful quality assessment techniques are needed in order to determine how well a system performs relative to a benchmark test, or how favorably it compares with a previous edition of the system or with an other designer's product. To the extend that a system performs less than perfect (something of which the author is aware), the designer will have to learn which aspect(s) and/or component(s) of his system are flawed

The needs of buyers and end users are different than those of designers but they too heavily rely on assessment techniques. Prospective buyers will always have a specific use of their TTS system in mind. Understandably, they will want the simplest, and therefore cheapest, system that satisfies their needs. The buyer will therefore need an absolute yardstick in order to determine beforehand if the TTS system is good enough to get a message across in the given application.

1.1. Taxonomy of Evaluation Tasks and Techniques

To justify our selections for the evaluation strategy used for the quality assessment of our TTS system, we will first discuss a number of distinguished parameters and explain the relationships between them.

The diagram shown in Figure 26 illustrates the relationships between the various dichotomies in the hierarchical order in which they have been listed in this diagram. Any path from the root down to any terminal that does not cross a horizontal gap constitutes a meaningful combination of test attributes.

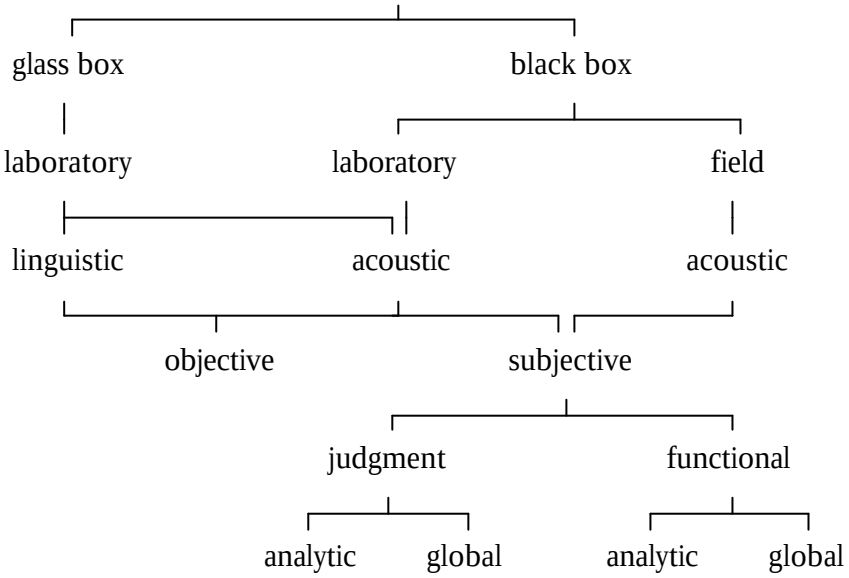

Figure 26 Relationships among dimensions involved in taxonomy of speech output evaluation methods

1.1.1. Black Box (Monolithic) versus Glass Box (Modular)

TTS systems generally comprise of a range of modules that take care of specific tasks (e.g. concatenation, signal processing).

End users will typically be interested in the performance of a system as a whole. They will consider the system as a *black box* that accepts text and outputs speech, a

monolith without any internal structure, since the quality of the output speech is the only thing that matters.

However if the output is less than optimal, it is almost impossible to pinpoint a certain cause for this malfunction. Therefore, for diagnostic purposes, designers often set up their evaluations in a more experimental (glass box) way. Keeping the effects of all modules but one constant, while systematically varying the characteristics of the latter, allows for any difference in the assessment of the system to be attributed to the variations of the target module.

1.1.2. Laboratory versus Field

TTS systems are often part of a human-machine user interface in a specific application. Typically, the vocabulary and types of information exchanges are restricted and domain specific, so that situational redundancy can often make up for bad intelligibility. On the other hand, TTS systems will often be used in complex information processing tasks, so that the listener has only limited resources available for attending to the speech input.

It is generally impossible to predict beforehand, on the basis of *laboratory tests*, exactly how successful a TTS-system will be in the practical application. The system needs to be tested in the *field*, i.e. in the real application, with real users. However, the use of *field tests* is limited to one system in one specific application; results of the test cannot, as a rule, be generalized to other systems and/or other applications.

1.1.3. Linguistic versus Acoustic

Complex TTS systems can roughly be divided into a linguistic interface that transforms spelling into an abstract phonological code and an acoustical interface that transduces this symbolic representation to an audible waveform.

The quality of the intermediary representation can be tested directly at the *symbolic-linguistic* level or indirectly at the level of the *acoustic* output. Testing the audio has the advantage that only errors in the symbolic representation that affect the audio output will affect the evaluation. However, it concerns human listeners and is therefore

costly and time consuming. Moreover the designer is not informed on the origin of any problems (linguistic or acoustic).

As an alternative, the intermediate representations in the linguistic interface are often evaluated in the symbolic level. It involves the comparison of the symbolic output of the linguistic model to a pre-stored model representation. The non-trivial problem is to obtain this model representation, which will have to be compiled manually, and will often involve multiple correct solutions.

1.1.4. Subjective versus Objective measurement

When an assessment technique involves the responses of human subjects, the measurement is called *subjective.*

It is most common that human subjects are called upon in order to evaluate the quality of a TTS system. This is to be expected, since the end user of a TTS system is a human listener. However there are certain drawbacks inherent to the use of human subjects. Firstly, humans are often somewhat noisy in their judgments, i.e. the results of tests are never perfectly reproducible. It often makes sense to use an expert listener as a shortcut to a preliminary evaluation, since he will be able to determine in great accuracy problems related to coarticulation, temporal organization and intonation. However he will not be able to predict in numerical terms how well the TTS system would perform as a communication tool with naïve listeners. Since this is what we need to assess, expert listeners should be used during the initial stages of development, as a design tool, while non-expert users should be used for the final evaluation of the system. In this case, a group of users may be used, and the average of their responses could somewhat compensate for the noisiness of their measurements. This is what is called *inter-subjective* measurement.

In addition to yielding noisy measurements, quality tests involving human listeners are also time consuming and therefore expensive to run. Automatic quality assessment for TTS systems that automatically measure the discrepancy in acoustical terms between a system's output and its human model is still a field under investigation. This is the type of *objective* evaluation technique that one would ultimately want to

come up with, since it avoids the use of human listeners, providing perfectly reproducible results in as little time as needed to run that particular test program. Unfortunately, these types of services are not yet available for use.

1.1.5. Judgment versus Functional

By *judgment* testing we mean a procedure whereby a group of listeners is asked to judge the performance of a TTS system, along a number of rating scales. The scales are typically bi-polar adjectives that allow the listeners to express the quality of the system.

A TTS system may also be assessed in terms of how well it actually performs its communicative purpose. This is called *functional* testing. For instance, if we want to know to what extent the output speech is intelligible, we may measure its intelligibility not by asking the listener how intelligible he things it is, but by determining, for instance, whether the listener correctly identifies the sounds.

1.1.6. Global versus Analytic

Judgment test usually include one or more rating scales covering such *global* aspects as "overall quality", "naturalness" and "acceptability".

On the other hand, one may be interested in determining the quality of specific aspects of a TTS system, in an *analytic* listening mode, where listeners are requested to pay particular attention to selected aspects of the speech output.

2. EVALUATION OF ACOUSTIC ASPECTS

Due to the nature of our synthesis algorithm (i.e. use of whole word units rather than phone segments), testing at the linguistic level is trivial, and has been used only during the design stage of our system.

The quality assessment of our TTS system was solely based on the evaluation of its acoustic aspects.

2.1. Aspects of Speech to be evaluated

There are three layers that are distinguished in speech: a segmental layer, (related to short-term fluctuation in the speech signal), a voice dynamics or prosodic layer (medium term fluctuations), and a voice quality layer (long term fluctuations).

We will make the same distinction in the evaluation of acoustic aspects.

2.1.1. Segments

The primary function of segments is simply to enable listeners to identify words. The segments used in our system are whole word units. In addition to that, we are penalizing the use of abnormally pronounced words during the synthesis of a sentence through their characterization with the reduction property. We may claim that this characterization actually constitutes a method of segment quality assessment. Thus, the question of word identification in the domain of our TTS system has been addressed only as part of the system's assessment in analytic listening mode.

2.1.2. Prosody

By prosody we mean the ensemble of properties of speech utterances that cannot be derived in a straightforward fashion from the identity of the phonemes constituting the words of the speech utterance. Prosody comprises the melody of the speech, word and phrase boundaries, word stress, sentence accent, tempo and changes in speaking rate.

The more important functions of prosody are located at the linguistic levels above the word:

- prosody tells the listener which words go together and should be interpreted as making up a coherent chunk of information; it also allows the user to determine whether he has come to the end of a word group, clause, sentence, etc.
- prosody provides an indication for the listener which words are presented by the speaker as expressing important information

- prosody, especially melody, caries its own intonational meaning, allowing for instance the speaker to present a sentence as a statement or a question

These observations suggest that prosody affects comprehension, which is what most functional tests of prosody try to evaluate.

2.1.3. Voice Quality

Voice quality can be viewed as the background against which segmental and prosodic variation is produced and perceived. It is used by the listener to form a (sometimes incorrect) idea of the speaker's mood and personality, physical size, sex, and also to identify the speaker. This information may have practical consequences for the continuation of the communication procedure, since it may influence the listener's attitude towards the speaker in a positive or negative sense, and may affect the listener's interpretation of the message.

2.1.4. Overall Output Quality

In most situations good intelligibility of specific words is not enough for TTS output to be called functionally adequate. One would want to have at one's disposal a functional test to evaluate the adequacy if the complete TTS output in all respects. In practice, the functional quality of overall TTS output has been equated with comprehension, based upon the integration of "bottom-up" speech signal information at different levels (segments, prosody, voice quality) and "top-down" knowledge and expectations based on previous experience, specific properties of the extra-linguistic context, and word internal and word combinatory redundancy.

2.2. Test Method

The importance of application specific test materials has been stressed by ITU-T's standardization sector. They developed a test specifically aimed at evaluating the quality of telephone speech, and which has been modified to fit our purposes. It is a judgment test comprising rating on eight scales, namely one 2-point scale *acceptance* and seven 5-point scales *overall impression*, *listening effort*, *comprehension problems*, *articulation*, *pronunciation*, *speaking rate*, and *voice pleasantness*.

Strictly speaking, only the first four scales can be captured under the heading *overall quality*; the other four scales are directed at more specific aspects of the output and require analytic listening. The content of the speech samples are synthesized in accordance with the application.

We used the ITU-T speech quality test for the evaluation of the output of our application. For the purposes of the test, nine weather forecast reports, compatible with the syntax of the EMY weather reports, were synthesized using several combinations of the selection criteria.

The synthesized utterances of these reports were organized in groups, in such a way that each group contained one weather report synthesized with each of these combinations. That means that each group consisted of nine weather reports, each one synthesized in a different manner, and put together in different order. Two people evaluated each such group. We asked the subjects to listen to a whole report, and then evaluate the quality of the synthetic speech using the eight scales we mentioned above. For the first scale (acceptance), the evaluation should determine whether the synthesized speech is accepted or not. For the rest seven scales, the evaluation was done by assigning a grade in the range of [0, 4], with 4 denoting the best performance.

The evaluations for each method among all groups were averaged, providing a measure for the performance of the method in each of these eight scales.

3. RESULTS

The evaluation procedure consisted of two stages. In the first stage we wanted to evaluate the performance of any combination of the selection criteria. In the second stage, we wanted to determine the improvement of the speech quality gained by introducing DSP methods to the synthesis strategy that performed best in the previous stage.

3.1. First Stage

There are four[3] criteria used for the selection of the segments to be concatenated in forming the desired synthesized utterance:

reduction
concatenation
position
coarticulation

The first criterion actually prevents the use of "oddly pronounced" words, and only occasionally does it affect the quality of the synthesized speech. So, the following synthesis strategies, utilizing combinations of the aforementioned criteria were used for the synthesis of the evaluation sentences:

1. None (reduction)
2. Pos (reduction + position)
3. Concat (reduction + concatenation)
4. Cooc (reduction + coarticulation)
5. Pos+Concat (reduction + position + concatenation)
6. Concat+Cooc (reduction + concatenation + coarticulation)
7. Pos+Cooc (reduction + position + coarticulation)
8. All (reduction + position + concatenation + coarticulation)
9. PreRec (prerecorded Corpus sentences)

As we can see, all strategies except 1) include Reduction in the selection criteria. Reduction by itself produces sentences with almost awful quality. On the other hand, the inclusion of prerecorded sentences, forming a meaningful Weather Report, is necessary in order to determine what the users believe is the optimum performance. A feature score of 2 may be considered average, but it is not that bad when the prerecorded utterances received a score of 3 for the same feature.

[3] There is a fifth one (modality) that was not used in the context of Weather Forecast Reports.

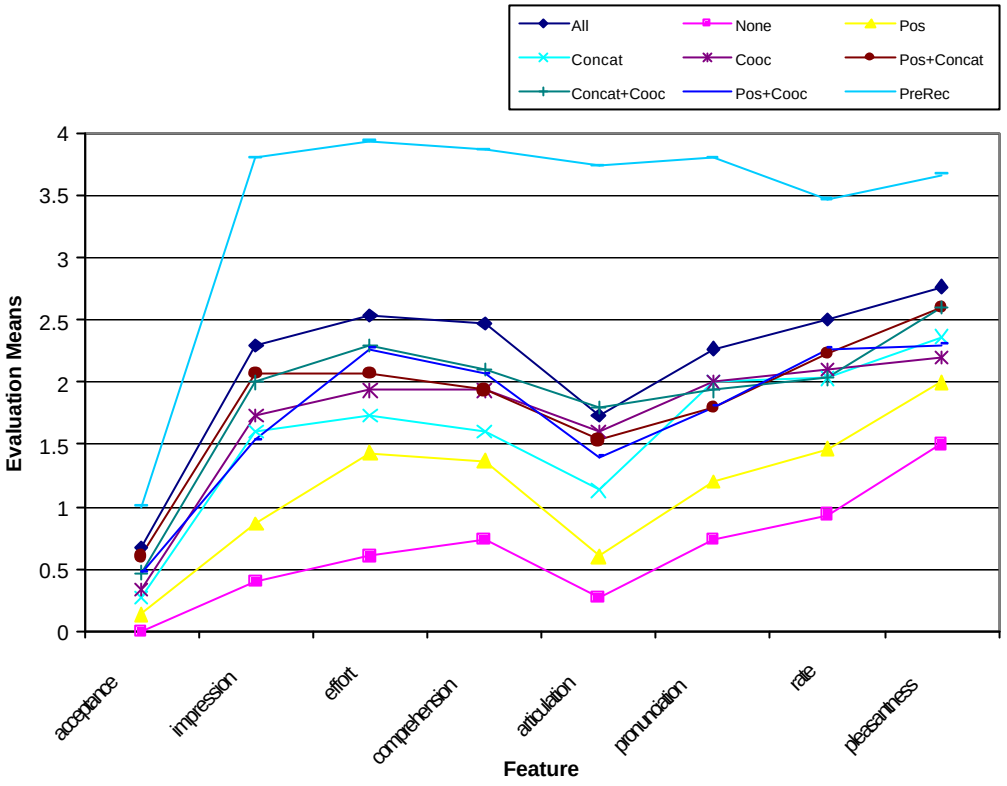

Figure 27 Evaluation Results (First Stage)

Figure 27 provides an overview of the evaluated performance for all the synthesis methods described above. As we may see, the forecasts that consisted of prerecorded utterances (PreRec) received the highest scores, followed by the combination of all selection criteria (All), which consistently receive the second best scores for all features. The worst performance, as expected, is observed for the sentences that used only the reduction criterion (None), since the choice of fragments used for the synthesis is almost random.

We will discuss the results for each feature.

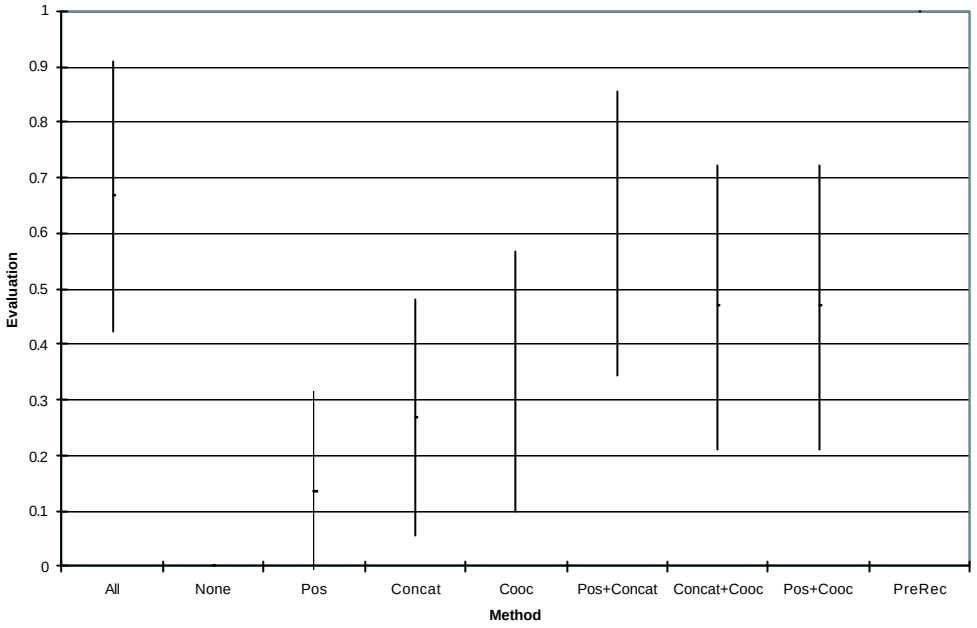

Figure 28 Acceptance

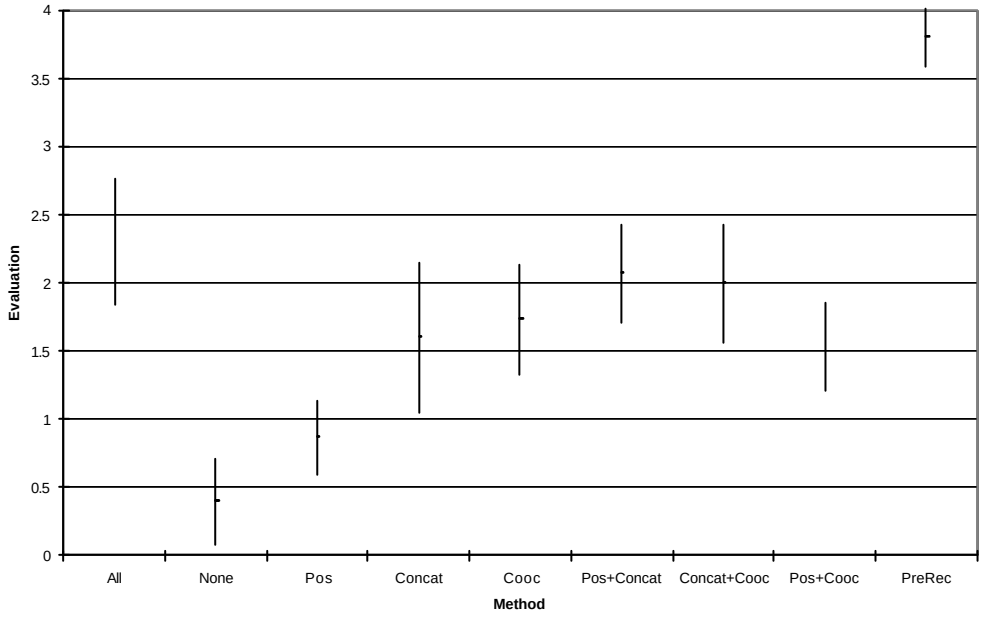

Figure 29 Overall Impression

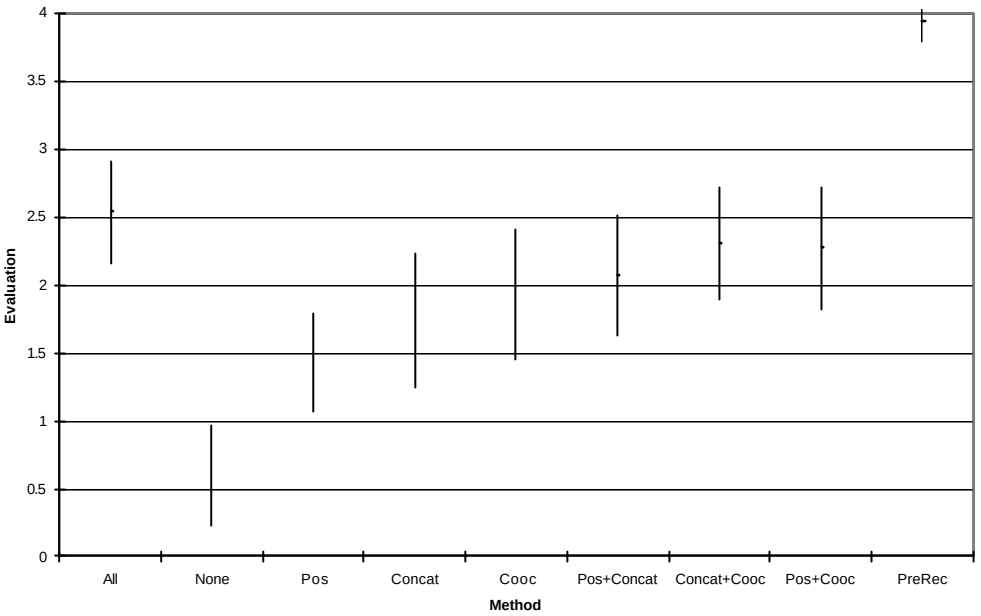

Figure 30 Listening Effort

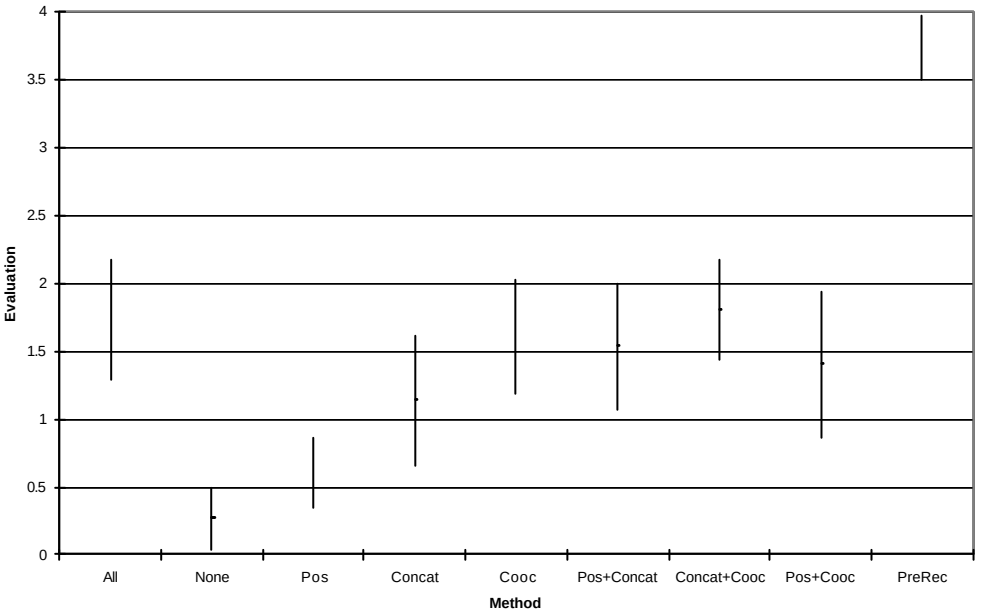

Figure 31 Comprehension Problems

The majority of the subjects have found it quite difficult to comprehend the message of the synthesized utterances. They all indicated that the greatest problems were erratic speech, differences in the volume of different words in the same sentence, inconsistent with the message of the sentence, and concatenation noises among word fragments.

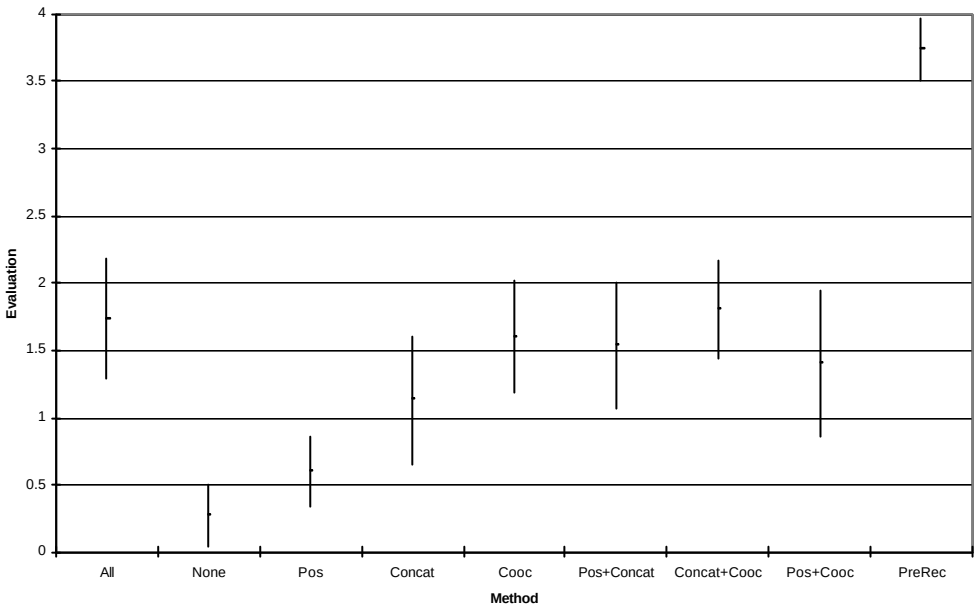

Figure 32 Articulation Problems

We may see that articulation poses the greatest challenge for the performance of our synthesizer. Methods utilizing co-occurrence information through the Coarticulation Cost, perform better than methods not using this information. However there exists a large gap between the best performing methods (All and Concat+Cooc) and the prerecorded utterances. We attribute this behavior to the absence[4] of energy smoothing at the edges of concatenated words.

[4] We remind the reader that the first stage **does not** include energy smoothing and energy averaging operations on the fragments used for utterance synthesis.

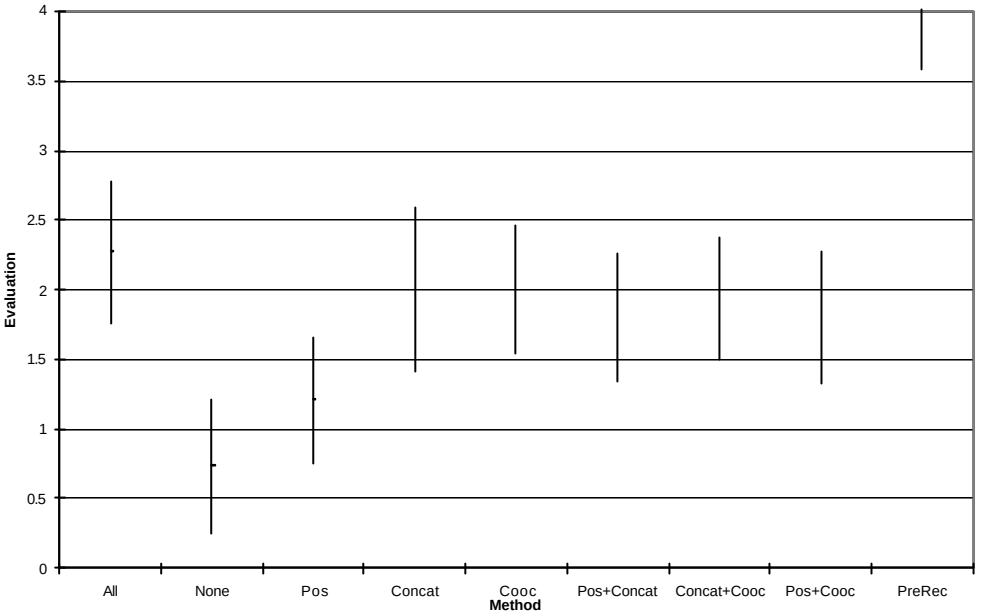

Figure 33 Pronunciation

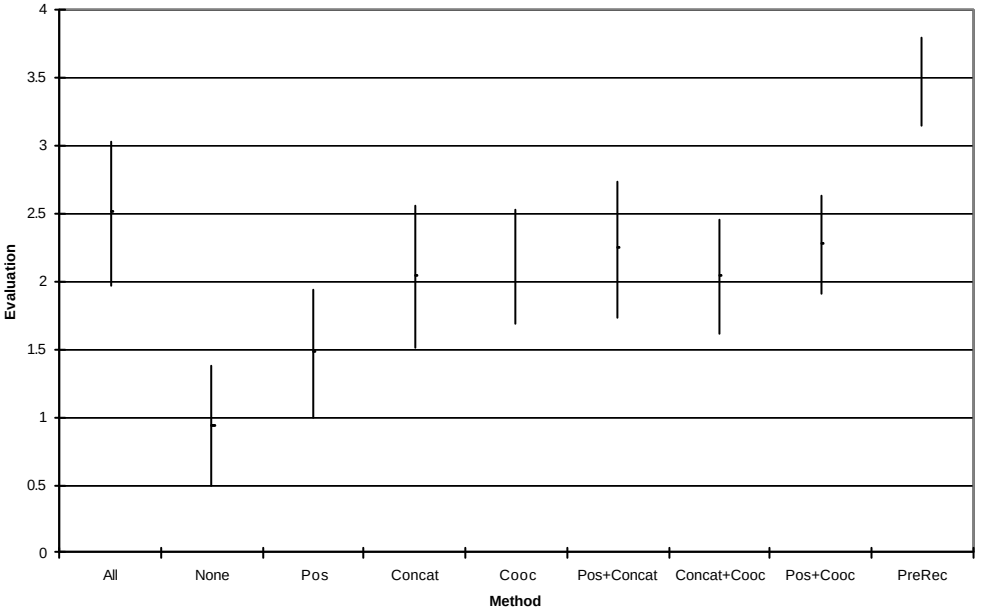

Figure 34 Speaking Rate

The fact that the prerecorded utterances did not receive as good a score as in other cases indicates that the subjects did not believe that the speaker has spoken these sentences in a totally accepted way. This problem has propagated itself in the rest methods as well.

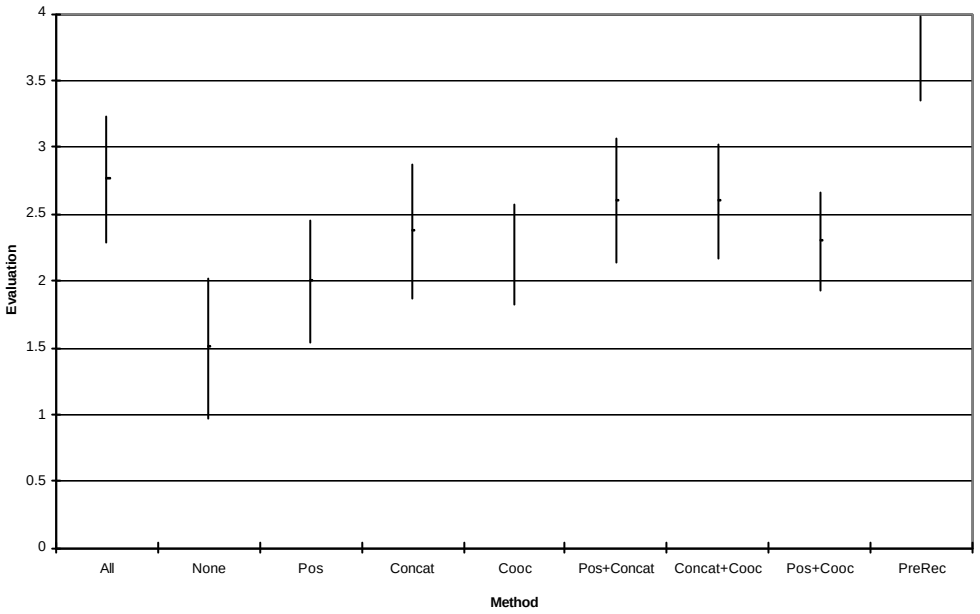

Figure 35 Voice Pleasantness

The same observations as for the previous feature apply here as well.

3.2. Second Stage

The first stage of the evaluation procedure indicated that the noises heard between two words, noises that may be attributed to differences in the energy levels of the concatenated words, as well as the difference in the volume of words in the same sentence are mostly responsible for most comprehension and listening effort problems.

The second stage of the evaluation procedure attempts to quantify the contribution of DSP methods to the overall quality of the synthesized speech.

In order to get an objective measure of the performance of these methods, we have asked the subjects to listen to five weather forecasts, each synthesized with one of the following strategies:

None (reduction)

Cooc (reduction + coarticulation)

All (reduction + position + concatenation + coarticulation)

All+DSP (reduction + position + concatenation + coarticulation + DSP)

PreRec (prerecorded utterances)

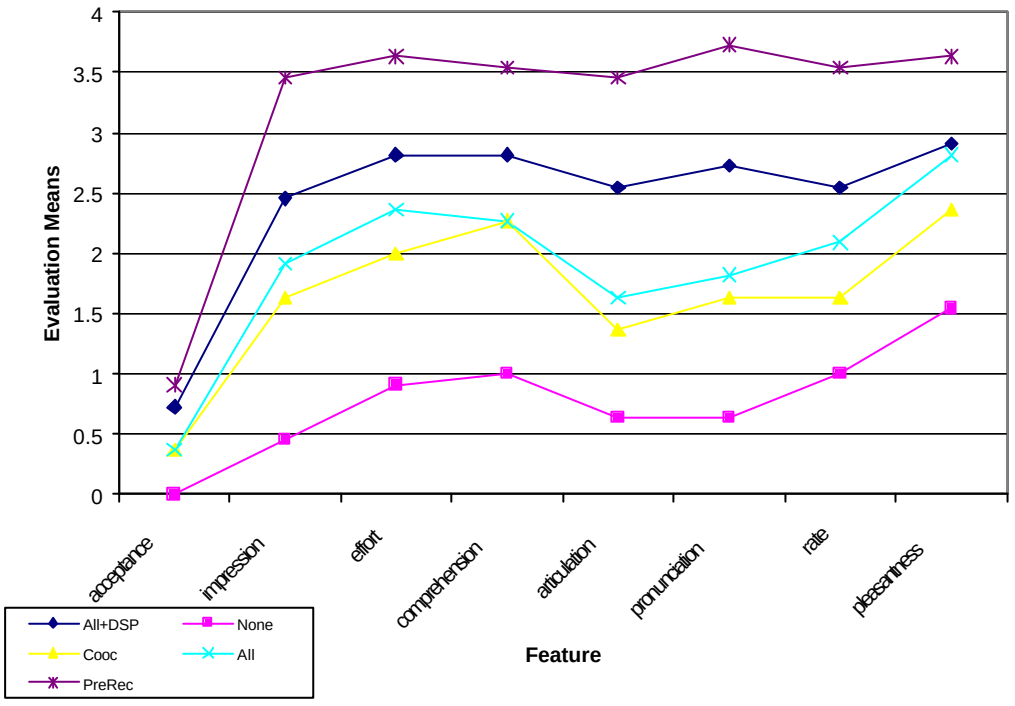

Figure 36 Evaluation Results (Second Stage)

We can see that the synthesis strategy that combines all the selection criteria along with DSP methods (All+DSP) has consistently outperformed the strategy that used only the selection criteria (All). This has lead to the creation of synthetic utterances that are even closer to natural speech, as one can notice by comparing the scores of (All+DSP) to those of (PreRec).

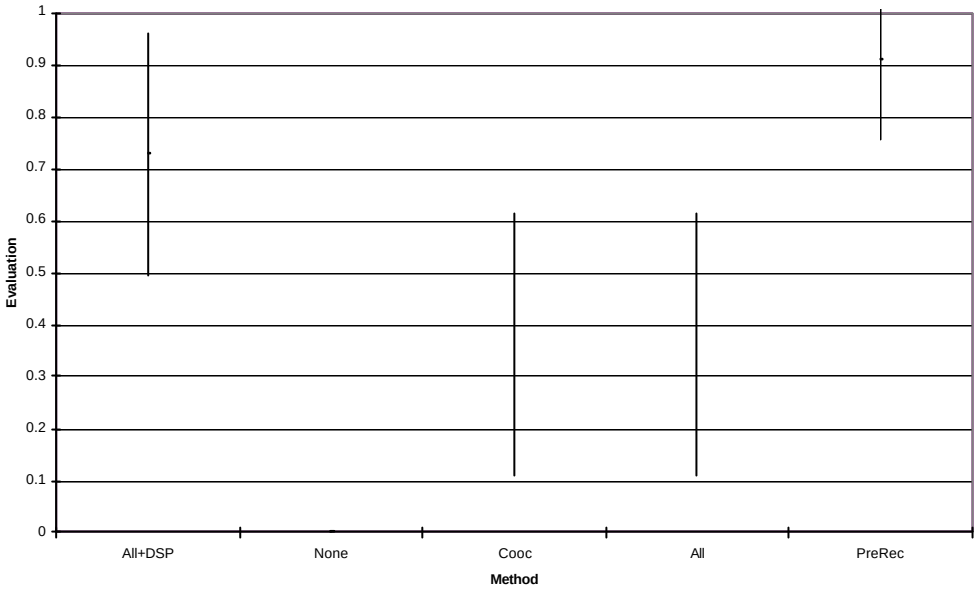

Figure 37 Acceptance

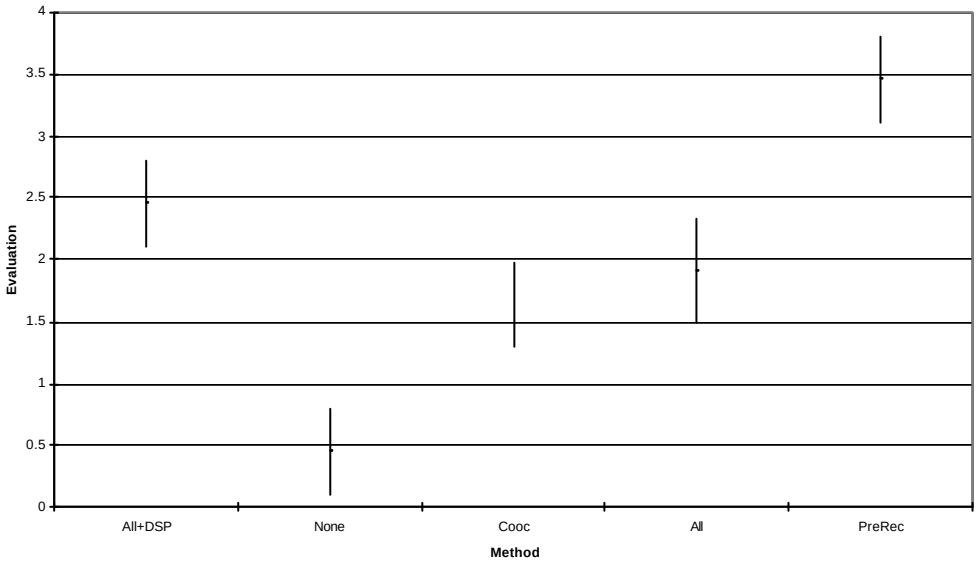

Figure 38 Overall Impression

In both of these charts, one may notice that not only has (All+DSP) outperformed (All), but that it has also made it less appealing to the subjects' evaluation.

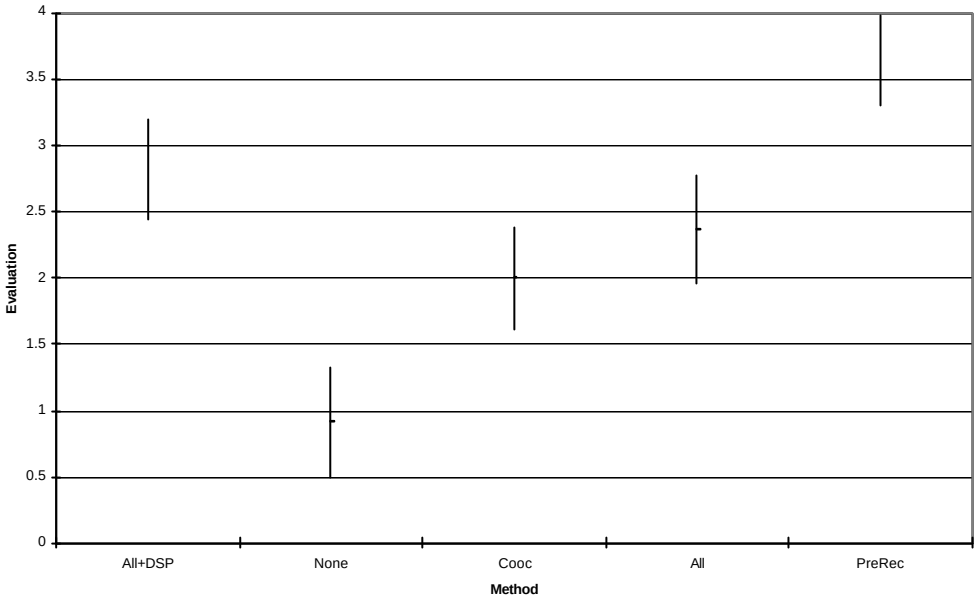

Figure 39 Listening Effort

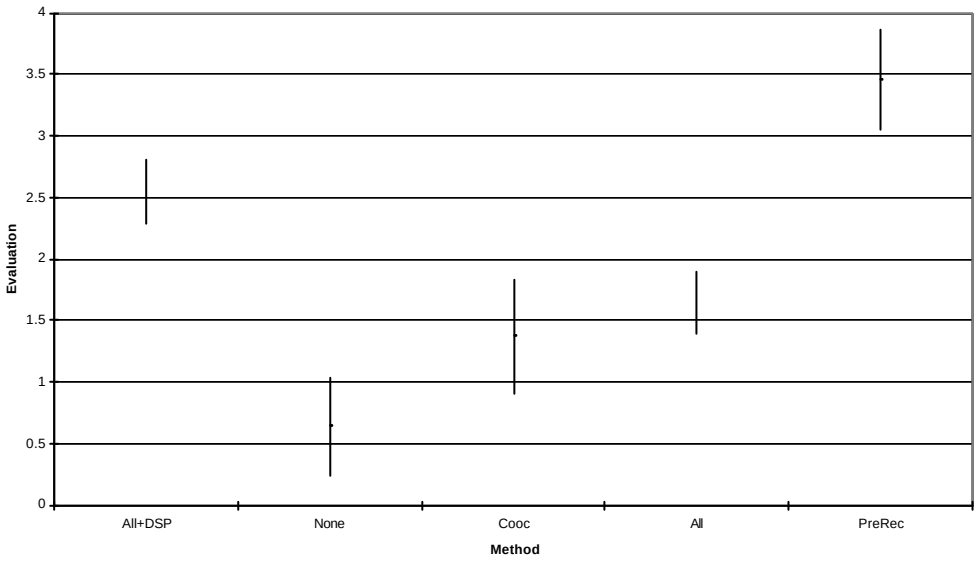

Figure 40 Comprehension Problems

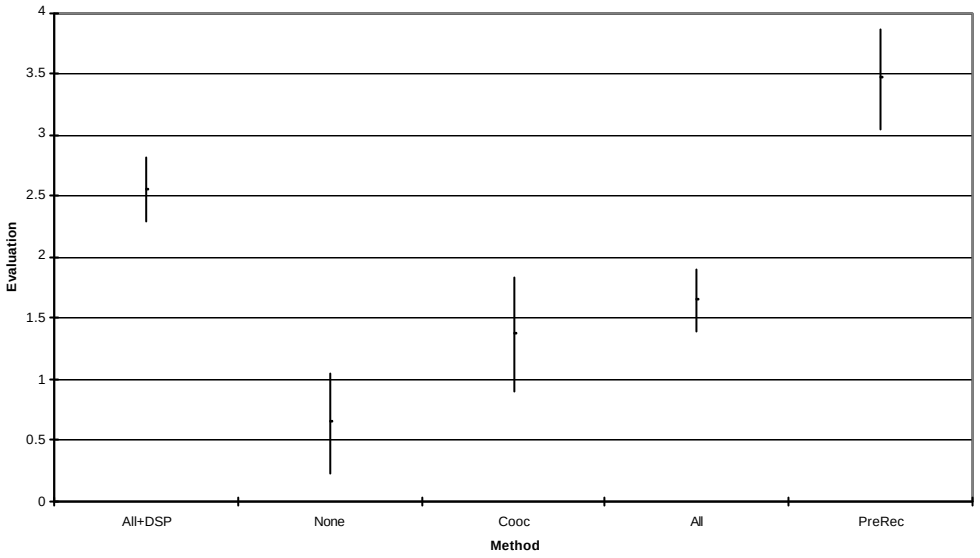

Figure 41 Articulation Problems

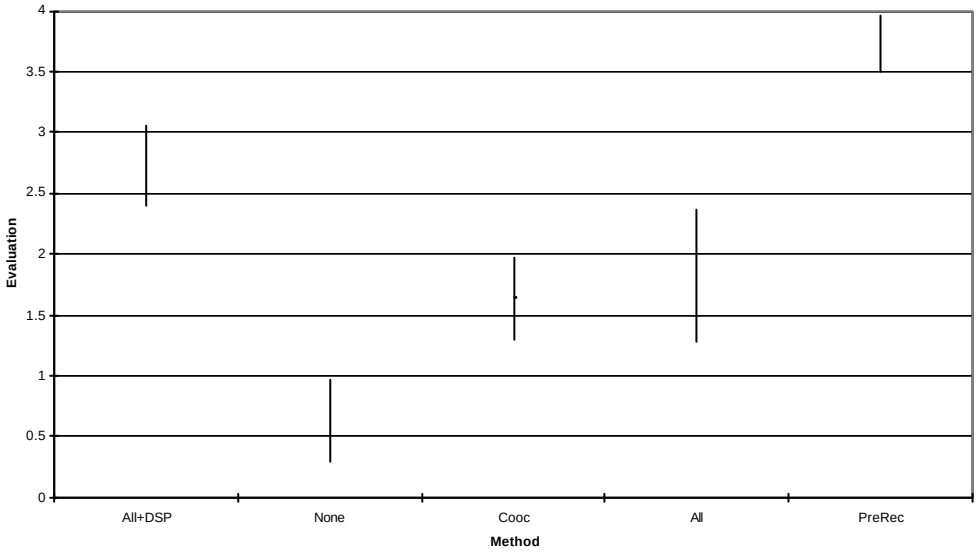

Figure 42 Pronunciation

The greatest differences between the two methods ((All) and (ALL+DSP)) is found in the evaluation of these two features. Smoothing the edges of concatenated fragments has reduced the severity of articulation problems. At the same time, the use

of fragments that were extracted from sentences that had all been normalized to the same average energy level has reduced unnecessary fluctuations in the volume of the sentence, thus rendering the synthesized output more “natural”. The same observation applies to Comprehension Problems and Listening Effort.

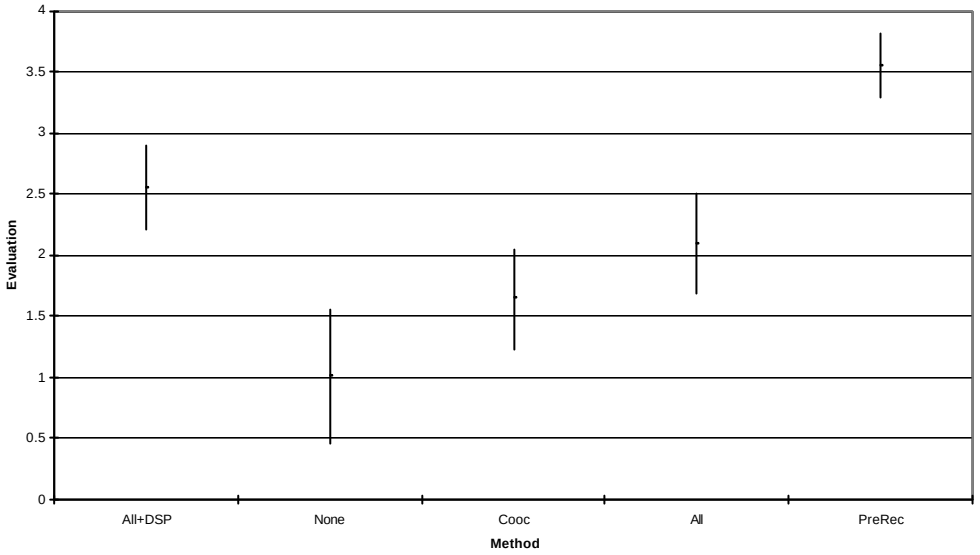

Figure 43 Speaking Rate

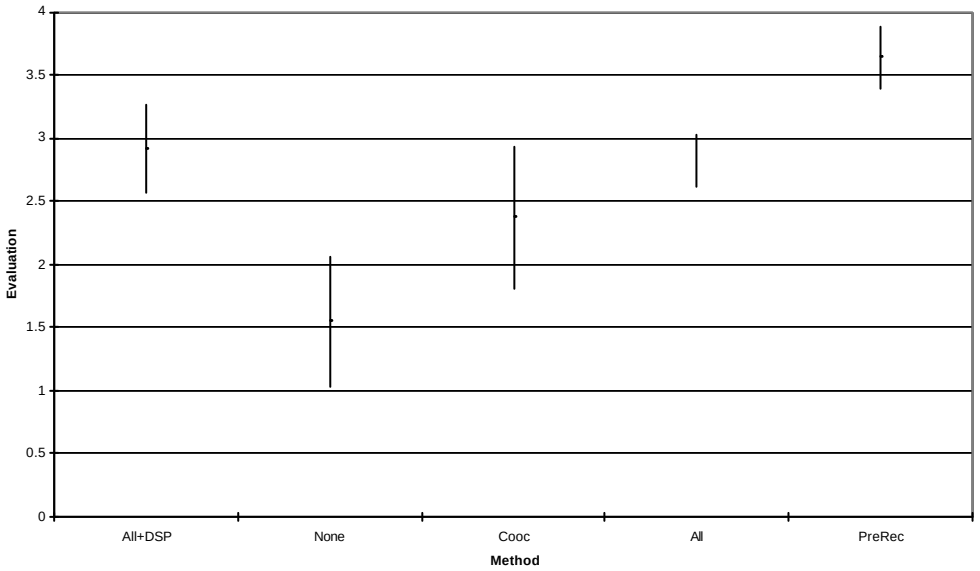

Figure 44 Voice Pleasantness

The fact that the Speaking Rate seems to be improved may be partially attributed to the smoothness of the volume of the sentence, something that most subjects regarded to be affecting the evaluation of this feature.

3.3 Conclusions

The following conclusions may be extracted from the evaluation procedure:

1. the Coarticulation cost is the single most significant feature in terms of quality
2. the combination of all features produces better results than any combination of up to three features
3. with the signal processing enhancements, the average acceptance score is 0.7, which compares favorably to the average score of 0.8 for the prerecorded sentences.

Chapter 6

FUTURE WORK

1. DISCUSSION

The talking computer HAL in the 1968 film "2001 – A Space Odyssey" had an almost human voice, but it was the voice of an actor, not a computer. Getting a real computer to talk like HAL has proven one of the toughest problems posed by "2001".

Many of the improvements in speech synthesis over the past five years have come from creative use of the technologies developed for speech recognition. We too have extensively used knowledge extracted from that field in order to improve the performance of our TTS system. The use of co-occurrence information for determining the coarticulation cost, information already used for the clustering of feature vectors in HMM-based SR systems, has greatly improved the quality of the synthesized speech. Automatic segmentation, aided by an SR system, has minimized the time needed to implement our system for domains other than the Weather Forecast domain.

Speech synthesis by word concatenation is a cheap, fast and simple way to do speech synthesis in restricted domains. Frequently the achieved quality is close to that one produced by humans. The only problem with such an approach is that storage complexity steeply rises as we try to cover wider domains.

2. FUTURE WORK

As a matter of course it is necessary to extend our approach to unrestricted domains. Therefore rules have to be developed which enable us to generate syllables from

phonemes and words from syllables. Thus, the system should perform unit concatenation in three different levels:

- diphones
- syllables
- words

The unit database should be modified to support such diverse units, while the selection process should adapt properly, choosing the best combination of units to synthesize the desired utterance.

Recent work on the creation of a language analysis for Greek in TUC, has provided us with the following facts:

Word #	Language Coverage
5,000	78.91%
10,000	85.32%
20,000	90.65%

Table 16 Vocabulary Size and Language Coverage for the Greek language

The data presented in Table 16 indicates that by extending the vocabulary size of our application to 20,000 WordClasses, there would only be one word out of ten in every sentence that would not be found in the Word Segment Database and that would need to be synthesized by simpler units (syllables, diphones, etc.). However, 20,000 WordClasses means that there would be approximately 100,000 Word Instrances, if the 1:5 ratio between WordClasses and Words (see also Table 3) stands for this application too.

It turns out that the storage complexity is much higher than that for diphone synthesis but this is not a real disadvantage. With the aid of signal processing it should be possible to reduce the number of recorded words as well as the number of stored samples. The number of stored samples may be easily reduced using compression algorithms. To reduce the number of words, additional research is required. In our

opinion, the question whether non-final sounding words might be transformed by signal manipulation into final sounding ones or vice versa is a main question of further work. Another interesting question is, whether it is possible to cluster instances of words so that only few prototypes need to be stored.

Our next step should extend the corpus annotation by phoneme segmentations based on manually corrected word boundaries. Together with automatically computed pitch marks, it would be possible to apply artificial F_0 and duration parameters using PSOLA manipulation to the synthetic signal.

Prominence is currently not explicitly considered in our selection criteria. It turns out to be the case that both the constrained domain of Weather Forecasts and our selection criteria already implicitly treat a number of prominence-related phenomena that need not be modeled by rule sets. Word class and prominence are highly correlated. This could explain the circumstance mentioned above. However, to respond to the necessities of Content-to-Speech (CTS), the generation of prosodic focus should be possible. For the planned extension of our synthesis using smaller units than words, prominence will play a major role. Therefore, an automatic labeling process should be developed which will mark the perceptual prominence of each unit.

3. EPILOGUE

The difference between a person and a talking computer is that the person understands the ideas and emotions conveyed through speech, and the computer doesn't. This is part of the larger problem of artificial intelligence, which is what "2001" author Arthur C. Clarke imagined in HAL. Our ability to replicate our own minds in a machine is limited by our incomplete knowledge of how our own minds work.

The ultimate goal for speech synthesis, as with all AI applications, is to make it pass the Turing Test - a blindfolded user shouldn't be able to tell whether he is talking to a human or a machine. Like the voice of HAL, that's a long way away. But we believe that modifying speech recognition techniques could lead to better speech synthesis

results. Ultimately the right model might just be the same for both synthesis and recognition.

APPENDICES

1. SINGLE-SOURCE SHORTEST PATHS

1.1. Background

The input to a weighted shortest path algorithm is a weighted directed graph $G = (V,E)$, with a weight function $w : E \rightarrow R$ mapping edges to real-valued weights. The **weight** of a path $p = \{v_1 v_2 \ldots v_N\}$ is

$$C(p) = \sum_{i=1}^{N-1} c_{i,i+1}$$

This is referred to as the **weighted path length**.

The **shortest path weight** from u to v is

$$d(u,v) = \begin{cases} \min\left\{w(p) : u \xrightarrow{p} v\right\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

The **shortest path** from vertex u to vertex v is then defined as any path p with weight

$$w(p) = d(u,v)$$

1.1.1. Representing Shortest Paths

We often wish to compute not only shortest path weights, but the vertices on the shortest paths as well. The representation we use for shortest paths is similar to the one used for breadth first trees.

Given a graph $G = (V,E)$, we maintain for each vertex $v \in V$ a ***predecessor*** $p[v]$ that is either a vertex of NIL.

A shortest paths algorithm sets the p attributes so that the chain of predecessors originating at vertex v runs backwards along a shortest path from s to v. Thus given a vertex v for which $p[v] \neq$ NIL, the shortest path from s to v can easily be tracked.

1.1.2. Relaxation

The technique of ***relaxation*** is used by Dijkstra's algorithm. For each vertex $v \in V$, we maintain an attribute $d[v]$, which is the upper bound on the weight of a shortest path from source s to v. We call $d[v]$ a ***shortest path estimate.*** We initialize the shortest path estimates and predecessors by the following procedure.

```
Initialize-Single-Source(G, s)
1 for each vertex v ∈ V[G]
2    do d[v] ← ∞
3       p[v] ← NIL
4 d[s] ← 0
```

After initialization, $p[v] =$ NIL for all $v \in V$, $d[v] = 0$ for $v = s$, and $d[v] = \infty$ for $v \in V - \{s\}$.

The process of ***relaxing***[5] an edge (u, v) consists of testing whether we can improve the shortest path to v so far by going through u, and, if so, updating $d[v]$ and $p[v]$. A relaxation step may decrease the value of the shortest path estimate $d[v]$ and update v's predecessor field $p[v]$. The following code performs a relaxation step on edge (u, v).

```
Relax(u, v, w)
1 if d[v] > d[u] + w(u,v)
```

[5] It may seem strange that the term "relaxation" is used for an operation that tightens an upper bound. The use of the term is historical. The outcome of a relaxation step can be viewed as a relaxation of the constraint $d[v] \leq d[u] + w(u, v)$, which must be satisfied if $d[u] = d(s, u)$ and $d[v] = d(s, v)$. That is, if $d[v] \leq d[u] + w(u, v)$, there is no "pressure" to satisfy this constraint, so the constraint is "relaxed".

```
2     then d[v] ? d[u] + w(u,v)
3          p[v] ? u
```

1.2. Dijkstra's Algorithm

Dijkstra's algorithm solves the single source shortest-paths problem on a weighted, directed graph $G = (V,E)$ for the case in which all edge weights are nonnegative. In this section, therefore, we assume that

$$w(u,v) \geq 0, \text{ for each edge } (u,v) \in E$$

Dijkstra's algorithm maintains a set S of vertices whose final shortest path weights from the source s have already been determined. That is

$$\forall v \in S \rightarrow d[v] = d(s,v)$$

The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest path estimate, inserts u into S, and relaxes all edges leaving u. In the following implementation we maintain a priority queue Q that contains all the vertices in $V - S$ keyed by their d values. The implementation assumes that graph G is represented in adjacency lists.

```
Dijkstra(G,w,s)
1 Initialize-Single-Source(G,s)
2 S ? Ø
3 Q ? V[G]
4 while Q ? Ø
5    do u ? Extract-Min(Q)
6       S ? S ∪ {u}
7       for each vertex v ∈ Adj[u]
8           do Relax(u,v,w)
```

Dijkstra's algorithm relaxes edges as shown in the pseudocode seen above. Line 1 performs the usual initialization of d and p values, and line 2 initializes the set S to the

empty set. Line 3 then initializes the priority queue Q to contain all the vertices in $V - S = V - \emptyset = V$.

Each time through he **while** loop of lines 4 – 8, a vertex u is extracted from $Q = V - S$ and inserted into set S. (The first time through the loop, $u = s$.) Vertex u, therefore, has the smallest shortest-path estimate of any vertex in $V - S$. Then, lines 7 – 8 relax each edge (u, v) leaving u, thus updating the estimate $d[v]$ and the predecessor $p[v]$ if the shortest path to v can be improved by going through u. Observe that vertices are never inserted into Q after line 3 and that each vertex is extracted from Q and inserted into S exactly once, so that the **while** loop of lines 4 – 8 iterates exactly $|V|$ times.

Because Dijkstra's algorithm always chooses the "lightest" or the "closest" vertex in $V - S$ to insert into set S, we say that it uses a greedy strategy. Greedy strategies do not always yield optimal results in general, but as the following theorem and its corollary show, Dijkstra's algorithm does indeed compute shortest paths. The key is to show that each time a vertex u is inserted into set S, we have $d[u] = d(s,u)$.

***Theorem*: Correctness of Dijkstra's Algorithm**

If we run Dijkstra's algorithm on a weighted, directed graph $G = (V,E)$ with nonnegative weight function w and source s, then at termination,

$$d[u] = d(s,u) \text{ for every vertex } u \in V$$

Corollary

If we run Dijkstra's algorithm on a weighted, directed graph $G = (V,E)$ with nonnegative weight function w and source s, then at termination, the predecessor subgraph G_p is the shortest-paths tree rooted at s.

1.2.1. Analysis

How fast is Dijkstra's algorithm? Consider first the case in which we maintain the priority queue $Q = V - S$ as a linear array. For such an implementation, each *Extract-*

Min operation takes time $O(V)$, and there are $|V|$ such operations, for a total *Extract-Min* time of $O(V^2)$. Each vertex $v \in V$ is inserted into set S exactly once, so each edge in the adjacency list $Adj[v]$ is examined in the **for** loop of lines 4 – 8 exactly once during the course of the algorithm. Since the total number of edges in all the adjacency lists in $|E|$, there are a total of $|E|$ iterations of this **for** loop, with each iteration taking $O(1)$ time. The running time of the entire algorithm is thus $O(V^2+E) = O(V^2)$.

If the graph is sparse, however, it is practical to implement the priority queue Q with a binary heap. The resulting algorithm is sometimes called the ***modified Dijkstra algorithm.*** Each *Extract-Min* operation then takes time $O(\log V)$. As before, there are $|V|$ such operation. The time to build the binary heap is $O(V)$. The assignment $d[v] = d[u] + w(u,v)$ in *Relax* is accomplished by the call *Decrease-Key*$(Q,v, d[u] + w(u,v))$, which takes time $O(\log V)$, and there are still at most $|E|$ such operations The total running time is therefore $O((V+E)\log V)$, which is $O(E\log V)$ if all vertices are reachable from the source.

We can in fact achieve a running time of $O(V\log V+E)$ by implementing the priority queue Q with a Fibbonacci heap. The amortized cost of each of the $|V|$ *Extract-Min* operations is $O(\log V)$, and each of the $|E|$ *Decrease-Key* calls takes only $O(1)$ amortized time. Historically, the development of Fibbonacci heaps was motivated by the observation that in the modified Dijkstra algorithm there are potentially many more *Decrease-Key* calls than *Extract-Min* calls, so any method of reducing the amortized time of *Extract-Min* would yield an asymptotically faster implementation.

Dijkstra's algorithm bears some resemblance to both breadth-first search and Prim's algorithm for computing minimum spanning trees. It is like breadth-first search in that S corresponds to the set of black vertices in a breadth-first search; just as vertices in S have their final shortest-path weights, so black vertices in a breadth-first search have their correct breadth first distances. Dijkstra's algorithm is like Prim's algorithm in that both algorithms use a priority queue to find the "lightest" vertex outside a given set (the set S in Dijkstra's algorithm and the tree being grown in Prim's

algorithm), insert this vertex into the set, and adjust the weights of the remaining vertices outside the set accordingly.

2. RESOURCE INTERFACE FORMAT FILES

The preferred format for multimedia files is resource interchange file format (RIFF). The RIFF file I/O functions work with the basic buffered and unbuffered file I/O services. RIFF files can be opened, read, and written in the same way as other file types.

RIFF files use four-character codes to identify file elements. These codes are 32-bit quantities representing a sequence of one to four ASCII alphanumeric characters, padded on the right with space characters. The data type for four-character codes is **FOURCC**. The **mmioFOURCC** macro may be used to convert four characters into a four-character code. To convert a null-terminated string into a four-character code, the **mmioStringToFOURCC** function should be used.

The basic building block of a RIFF file is a *chunk*. A chunk is a logical unit of multimedia data, such as a single frame in a video clip. Each chunk contains the following fields:

- A four-character code specifying the chunk identifier
- A doubleword value specifying the size of the data member in the chunk
- A data field

The following illustration shows a "RIFF" chunk that contains two subchunks.

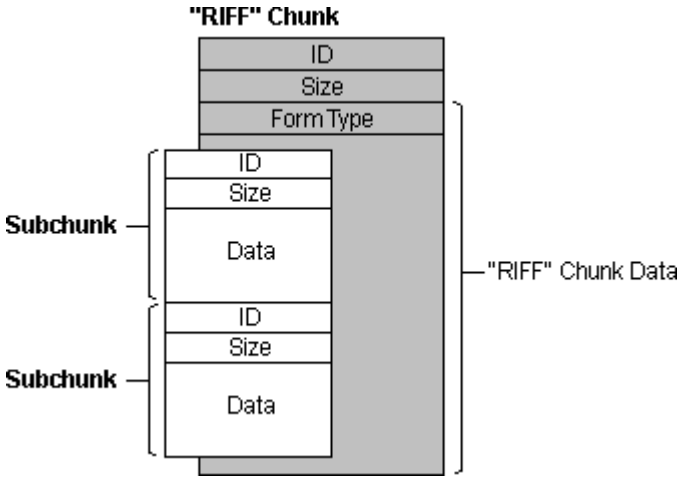

Figure 45 "RIFF" chunk containing two subchunks

A chunk contained in another chunk is a *subchunk*. The only chunks allowed to contain subchunks are those with a chunk identifier of "**RIFF**" or "**LIST**". A chunk that contains another chunk is called a *parent chunk*. The first chunk in a RIFF file must be a "RIFF" chunk. All other chunks in the file are subchunks of the "RIFF" chunk.

"RIFF" chunks include an additional field in the first four bytes of the data field. This additional field provides the *form type* of the field. The form type is a four-character code identifying the format of the data stored in the file. For example, Microsoft waveform-audio files have a form type of "WAVE".

"LIST" chunks also include an additional field in the first four bytes of the data field. This additional field contains the *list type* of the field. The list type is a four-character code identifying the contents of the list. For example, a "LIST" chunk with a list type of "INFO" can contain "ICOP" and "ICRD" chunks providing copyright and creation date information. The following illustration shows a "RIFF" chunk that contains a "LIST" chunk and one other subchunk (the "LIST" chunk contains two subchunks).

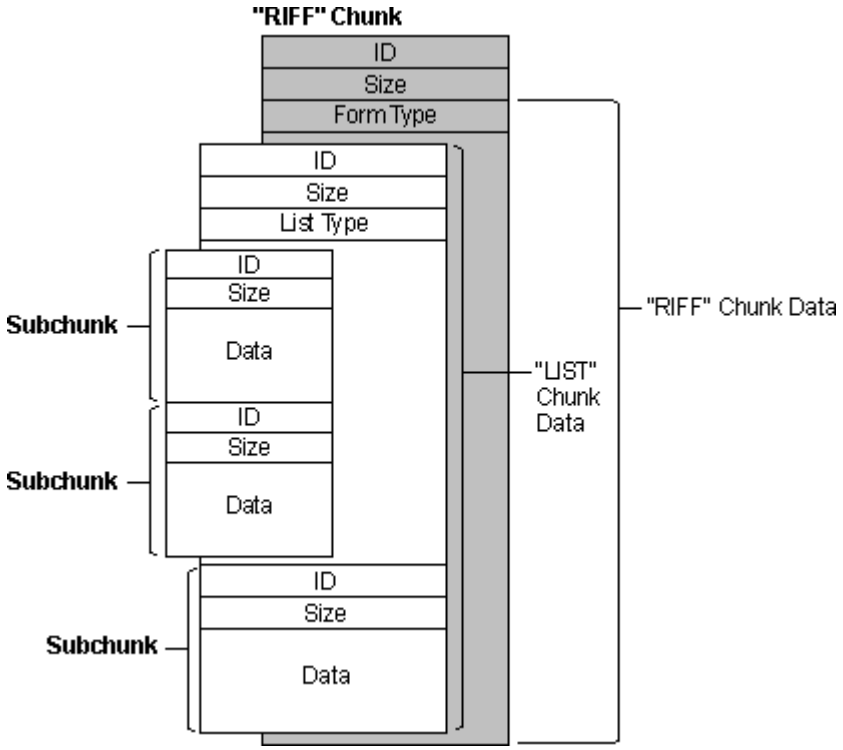

Figure 46 "RIFF" chunk containing a "LIST" subchunk

Multimedia file I/O services include two functions that can be used to navigate among chunks in a RIFF file: **mmioAscend** and **mmioDescend**. These functions can be used as high-level seek functions. When descending into a chunk, the file position is set to the data field of the chunk (8 bytes from the beginning of the chunk). For "RIFF" and "LIST" chunks, the file position is set to the location following the form type or list type (12 bytes from the beginning of the chunk). When ascending out of a chunk, the file position is set to the location following the end of the chunk.

To create a new chunk, the **mmioCreateChunk** function can be used to write a chunk header at the current position in an open file. The **mmioAscend**, **mmioDescend**, and **mmioCreateChunk** functions use the **MMCKINFO** structure to specify and retrieve information about "RIFF" chunks.

BIBLIOGRAPHY

Dutoit, Th. *An Itroduction to Text-to-Speech Synthesis*, Dordrecht, Kluwer Academic Publishers, 1997.

Weiss, M. A. *Data Structures & Algorithm Analysis in C++*, 2nd ed. Addison Wesley Longman, 1999.

Raman, T. V. *Auditory User Interfaces: Toward the Speaking Computer*, Dordrecht, Kluwer Academic Press, 1997

Cormen, T. H., Leiserson, C. E. and Rivest, R. R. *Introduction to Algorithms*, Cambridge MA, The MIT Press, 1990

Stöber, K., Portele, T., Wagner, P., Hess, W., "Synthesis By Word Concatenation", *Proceedings of EuroSpeech 1999, Budapest, Hungary*, Vol. 2:619-622, 1999

Lieberman, M., "Computer Speech Synthesis: Its Status and Prospects", *Voice Communication between Humans and Machines*, Washington D.C., National Academy Press, 1994

Sproat, R., Olive, J., "An Approach to Text-to-Speech Synthesis", *Speech Coding and Synthesis*, The Netherlands, Elsevier Science, 1998

Heuven, V. J. van, Bezooijven, R. van, "Quality Evaluation of Synthesized Speech", *Speech Coding and Synthesis*, The Netherlands, Elsevier Science, 1998

Moulines, E., Charpentier, F., "Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones", *Speech Communication*, 9(5/6):453-467, 1990

Allen, J., Hunnicut, S., Klatt, D., *From Text to Speech, The MITalk System*, Cambridge, Cambrdge University Press, 1987

Beutnagel, M., Mohri, M., Riley, M., "Rapid Unit Selection from a Large Speech Corpus for Concatenative Synthesis", *Proceedings of EuroSpeech 1999, Budapest, Hungary*, Vol. 2: 607-610, 1999

Hunt, A., Black, A., "Unit Selection in a Concatenative Speech Synthesis System using a Large Speech Database", *Proceedings of ICASSP-96, Atlanta, GA*, Vol. 1: 373-376, 1996

Lewis, E., Tatham, M., "Word and Syllable Concatenation in Text-to-Speech Synthesis", *Proceedings of EuroSpeech 1999, Budapest, Hungary*, Vol. 2: 615-618, 1999

Balestri, M., Pacchiotti, A., Quazza, S., Salza, P. L., Sandri, S., "Choose the Best to modify the Least: A New Generation Concatenative Synthesis System", *Proceedings of EuroSpeech 1999, Budapest, Hungary*, Vol. 5:2291-2294