XUEDONG HUANG | ALEX ACERO | HSIAO-WUEN HON

SPOKEN LANGUAGE PROCESSING

A Guide to Theory, Algorithm, and System Development

Foreword by Dr. Raj Reddy
Carnegie Mellon University

Spoken Language Processing

(b)

Spoken Language Processing

A Guide to Theory, Algorithm, and System Development

Xuedong Huang
Alex Acero
Hsiao-Wuen Hon

Microsoft Research

Prentice Hall PTR
Upper Saddle River, New Jersey 07458
www.phptr.com

Library of Congress Cataloging-in-Publication Data

Huang, Xuedong.
Spoken language processing: a guide to theory, algorithm, and system development/
Xuedong Huang, Alex Acero, Hsiao-Wuen Hon.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-022616-5
1. Natural language processing (Computer science) I. Acero, Alex. II. Hon,
Hsiao-Wuen. III. Title.

QA76.9.N38 H83 2001
006.3'5—dc21 00-050196

Editorial/production supervision: *Jane Bonnell*
Cover design director: *Jerry Votta*
Cover design: *Anthony Gemmellaro*
Manufacturing buyer: *Maura Zaldivar*
Development editor: *Russ Hall*
Acquisitions editor: *Tim Moore*
Editorial assistant: *Allyson Kloss*
Marketing manager: *Debby van Dijk*

Prentice-Hall, Inc.
Upper Saddle River, New Jersey 07458

Prentice Hall books are widely used by corporations and government agencies for training, marketing, and resale.
The publisher offers discounts on this book when ordered in bulk quantities. For more information, contact Corporate Sales Department, Phone: 800-382-3419; FAX: 201-236-7141;
E-mail: corpsales@prenhall.com
Or write: Prentice Hall PTR, Corporate Sales Dept., One Lake Street, Upper Saddle River, NJ 07458.

Printed in the United States of America

ISBN 0-13-022616-5

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Pearson Education Asia Pte. Ltd.
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

To Yingzhi, Angela, Christina, and Derek

To Donna and Nicolas

To Phen, Stephanie, and Jacqueline

Contents

PART III: SPEECH RECOGNITION

PART IV: TEXT-TO-SPEECH SYSTEMS

Foreword

Recognition and understanding of spontaneous unrehearsed speech remains an elusive goal. To understand speech, a human considers not only the specific information conveyed to the ear, but also the context in which the information is being discussed. For this reason, people can understand spoken language even when the speech signal is corrupted by noise. However, understanding the context of speech is, in turn, based on a broad knowledge of the world. And this has been the source of the difficulty and over forty years of research.

It is difficult to develop computer programs that are sufficiently sophisticated to understand continuous speech by a random speaker. Only when programmers simplify the problem—by isolating words, limiting the vocabulary or number of speakers, or constraining the way in which sentences may be formed—is speech recognition by computer possible.

Since the early 1970s, researchers at AT&T, BBN, CMU, IBM, Lincoln Labs, MIT, and SRI have made major contributions in Spoken Language Understanding Research. In 1971, the Defense Advanced Research Projects Agency (DARPA) initiated an ambitious five-year, $15 million, multisite effort to develop speech understanding systems. The goals were to develop systems that would accept continuous speech from many speakers, with minimal speaker adaptation, and operate on a 1000-word vocabulary, artificial syntax, and a

constrained task domain. Two of the systems, Harpy and Hearsay-II, both developed at Carnegie Mellon University, achieved the original goals and in some instances surpassed them.

During the last three decades I have been at Carnegie Mellon, I have been very fortunate to be able to work with many brilliant students and researchers. Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon were arguably among the outstanding researchers in the speech group at CMU. Since then, they have moved to Microsoft and have put together a world-class team at Microsoft Research. Over the years, they have contributed standards for building spoken language understanding systems with Microsoft's SAPI/SDK family of products and pushed the technologies forward with the rest of the community. Today, they continue to play a premier leadership role in both the research community and in industry.

This new book, *Spoken Language Processing*, represents a welcome addition to the technical literature on this increasingly important emerging area of Information Technology. As we move from desktop PCs to personal digital assistants (PDAs), wearable computers, and Internet cell phones, speech becomes a central, if not the only, means of communication between the human and machine! Huang, Acero, and Hon have undertaken a commendable task of creating a comprehensive reference that covers theoretical, algorithmic, and systems aspects of the spoken language tasks of recognition, synthesis, and understanding.

The task of spoken language communication requires a system to recognize, interpret, execute, and respond to a spoken query. This task is complicated by the fact that the speech signal is corrupted by many sources: noise in the background, characteristics of the microphone, vocal tract characteristics of the speakers, and differences in pronunciation. In addition, the system has to cope with non-grammaticality of spoken communication and ambiguity of language. An effective system must strive to utilize all the available sources of knowledge—acoustics, phonetics and phonology, lexical, syntactic, and semantic structure of language, and task-specific context-dependent information.

Speech is based on a sequence of discrete sound segments that are linked in time. These segments, called phonemes, are assumed to have unique articulatory and acoustic characteristics. While the human vocal apparatus can produce an almost infinite number of articulatory gestures, the number of phonemes is limited. English as spoken in the United States, for example, contains 16 vowel and 24 consonant sounds. Each phoneme has distinguishable acoustic characteristics and, in combination with other phonemes, forms larger units such as syllables and words. Knowledge about the acoustic differences among these sound units is essential to distinguish one word from another, say, *bit* from *pit*.

When speech sounds are connected to form larger linguistic units, the acoustic characteristics of a given phoneme will change as a function of its immediate phonetic environment because of the interaction among various anatomical structures (such as the tongue, lips, and vocal chords) and their different degrees of sluggishness. The result is an overlap of phonemic information in the acoustic signal from one segment to the other. For example, the same underlying phoneme *t* can have drastically different acoustic characteristics in different words, say, in *tea*, *tree*, *city*, *beaten*, and *steep*. This effect, known as coarticulation, can occur within a given word or across a word boundary. Thus, the word *this* will have very different acoustic properties in phrases such as *this car* and *this ship*.

This book is self-contained for those who wish to familiarize themselves with the current state of spoken language systems technology. However, a researcher or a professional in the field will benefit from a thorough grounding in a number of disciplines, including:

- *Signal processing*: Fourier Transforms, DFT, and FFT
- *Acoustics*: physics of sounds and speech, models of vocal tract
- *Pattern recognition*: clustering and pattern matching techniques
- *Artificial intelligence*: knowledge representation and search, natural language processing
- *Computer science*: hardware, parallel systems, algorithm optimization
- *Statistics*: probability theory, hidden Markov models, dynamic programming
- *Linguistics*: acoustic phonetics, lexical representation, syntax, and semantics

A newcomer to this field, easily overwhelmed by the vast number of different algorithms scattered across many conference proceedings, can find in this book a set of techniques that Huang, Acero, and Hon have found to work well in practice. This book is unique in that it includes both the theory and implementation details necessary to build spoken language systems. If you were able to assemble all the individual material that is covered in the book and put it on a shelf, it would be several times larger than this volume and yet you would be missing vital information. You would not have the material that is in this book that threads it all into one story, one context. If you need additional resources, the authors include extensive references to get that additional detail. *Spoken Language Processing* is very appealing both as a textbook and as a reference book for practicing engineers. Some readers familiar with a specific topic may decide to skip a few chapters; others may want to focus in other chapters. This is not a book that you will pick up and read once from cover to cover, but one you will keep near you for reference as long as you work in this field.

Raj Reddy
Dean, School of Computer Science
Carnegie Mellon University

Preface

Our primary motivation in writing this book is to share our working experience to bridge the gap between the knowledge of industry gurus and newcomers to the spoken language processing community. Many powerful techniques hide in conference proceedings and academic papers for years before becoming widely recognized by the research community or the industry. We spent many years pursuing spoken language technology research at Carnegie Mellon University before we started spoken language R&D at Microsoft. We fully understand that it is by no means a small undertaking to transfer a state-of-the-art spoken language research system into a commercially viable product that can truly help people improve their productivity. Our experience in both industry and academia is reflected in the context of this book, which presents a contemporary and comprehensive description of both theoretic and practical issues in spoken language processing. This book is intended for people of diverse academic and practical backgrounds. Speech scientists, computer scientists, linguists, engineers, physicists, and psychologists all have a unique perspective on spoken language processing. This book will be useful to all of these special interest groups.

Spoken language processing is a diverse subject that relies on knowledge of many levels, including acoustics, phonology, phonetics, linguistics, semantics, pragmatics, and discourse. The diverse nature of spoken language processing requires knowledge in computer science, electrical engineering, mathematics, syntax, and psychology. There are a number of excellent books on the subfields of spoken language processing, including speech recognition, text-to-speech conversion, and spoken language understanding, but there is no single book that covers both theoretical and practical aspects of these subfields and spoken language interface design. We devote many chapters systematically introducing fundamental

theories needed to understand how speech recognition, text-to-speech synthesis, and spoken language understanding work. Even more important is the fact that the book highlights what works well in practice, which is invaluable if you want to build a practical speech recognizer, a practical text-to-speech synthesizer, or a practical spoken language system. Using numerous real examples in developing Microsoft's spoken language systems, we concentrate on showing how the fundamental theories can be applied to solve real problems in spoken language processing.

We would like to thank many people who helped us during our spoken language processing R&D careers. We are particularly indebted to Professor Raj Reddy at the School of Computer Science, Carnegie Mellon University. Under his leadership, Carnegie Mellon University has become a center of research excellence on spoken language processing. Today's computer industry and academia benefit tremendously from his leadership and contributions.

Special thanks are due to Microsoft for its encouragement of spoken language R&D. The management team at Microsoft has been extremely generous to the speech technology group. We are particularly grateful to Bill Gates, Nathan Myhrvold, Rick Rashid, Dan Ling, and Jack Breese for the great environment they have created for us at Microsoft Research. We would also like to thank Bob Muglia and Kai-Fu Lee for their leadership role in Microsoft's speech product development.

Scott Meredith helped us write a number of chapters in this book and deserves to be a co-author. His insight and experience in text-to-speech synthesis enriched this book a great deal. We also owe gratitude to many colleagues we worked with in the speech technology group of Microsoft Research. In alphabetic order, Jim Adcock, Bruno Alabiso, Fil Alleva, Eric Bidstrup, Antonio Bigazzi, Ciprian Chelba, Li Deng, James Droppo, Doug Duchene, Joshua Goodman, Mei-Yuh Hwang, Larry Israel, Derek Jacoby, Li Jiang, Yun-Cheng Ju, David Larson, Kevin Larson, Jingsong Liu, Ricky Loynd, Milind Mahajan, Peter Mau, John Merrill, Yunus Mohammed, Salman Mughal, Mike Plumpe, Scott Quinn, Bill Rockenbeck, Mike Rozak, Kevin Schofield, Roxana Teodorescu, Gina Venolia, Kuansan Wang, Ye-Yi Wang, and Shenzhi Zhang.

In addition, we want to thank Les Atlas, Jeff Bilmes, Alan Black, David Caulton, Eric Chang, Phil Chou, Dinei Florencio, Allen Gersho, Francisco Gimenez-Galanes, Hynek Hermansky, Henrique Malvar, Julian Odell, Mari Ostendorf, Joseph Pentheroudakis, Tandy Trower, and Charles Wayne. They provided us with many wonderful comments to refine this book. Tim Moore, Russ Hall, and Jane Bonnell at Prentice Hall helped us finish this book in a finite amount of time.

Finally, writing this book was a marathon that could not have been finished without the support of our spouses, Yingzhi, Donna, and Phen, during the many evenings and weekends we spent on this project.

Xuedong Huang
Alex Acero
Hsiao-Wuen Hon
Redmond, WA

CHAPTER 1

Introduction

From human prehistory to the new media of the future, speech communication has been and will be the dominant mode of human social bonding and information exchange. The spoken word is now extended, through technological mediation such as telephony, movies, radio, television, and the Internet. This trend reflects the primacy of spoken communication in human psychology.

In addition to human-human interaction, this human preference for spoken language communication finds a reflection in human-machine interaction as well. Most computers currently utilize a *graphical user interface* (GUI), based on graphically represented interface objects and functions such as windows, icons, menus, and pointers. Most computer operating systems and applications also depend on a user's keyboard strokes and mouse clicks, with a display monitor for feedback. Today's computers lack the fundamental human abilities to speak, listen, understand, and learn. Speech, supported by other natural modalities, will be one of the primary means of interfacing with computers. And, even before speech-based interaction reaches full maturity, applications in home, mobile, and office segments are incorporating spoken language technology to change the way we live and work.

A spoken language system needs to have both speech recognition and speech synthesis capabilities. However, those two components by themselves are not sufficient to build a useful spoken language system. An understanding and dialog component is required to manage interactions with the user; and domain knowledge must be provided to guide the system's interpretation of speech and allow it to determine the appropriate action. For all these components, significant challenges exist, including robustness, flexibility, ease of integration, and engineering efficiency. The goal of building commercially viable spoken language systems has long attracted the attention of scientists and engineers all over the world. The purpose of this book is to share our working experience in developing advanced spoken language processing systems with both our colleagues and newcomers. We devote many chapters to systematically introducing fundamental theories and to highlighting what works well based on numerous lessons we learned in developing Microsoft's spoken language systems.

1.1. MOTIVATIONS

What motivates the integration of spoken language as the primary interface modality? We present a number of scenarios, roughly in order of expected degree of technical challenges and expected time to full deployment.

1.1.1. Spoken Language Interface

There are generally two categories of users who can benefit from adoption of speech as a control modality in parallel with others, such as the mouse, keyboard, touch-screen, and joystick. For novice users, functions that are conceptually simple should be directly accessible. For example, raising the voice output volume under software control on the desktop speakers, a conceptually simple operation, in some GUI systems of today requires opening one or more windows or menus, and manipulating sliders, check-boxes, or other graphical elements. This requires some knowledge of the system's interface conventions and structures. For the novice user, to be able to say *raise the volume* would be more direct and natural. For expert users, the GUI paradigm is sometimes perceived as an obstacle or nuisance and shortcuts are sought. Frequently these shortcuts allow the power user's hands to remain on the keyboard or mouse while mixing content creation with system commands. For example, an operator of a graphic design system for CAD/CAM might wish to specify a text formatting command while keeping the pointer device in position over a selected screen element.

Speech has the potential to accomplish these functions more powerfully than keyboard and mouse clicks. Speech becomes more powerful when supplemented by information streams encoding other dynamic aspects of user and system status, which can be resolved by the semantic component of a complete multimodal interface. We expect such multimodal interactions to proceed based on more complete user modeling, including speech, visual orientation, natural and device-based gestures, and facial expression, and these will be coordinated with detailed system profiles of typical user tasks and activity patterns.

In some situations you must rely on speech as an input or output medium. For example, with wearable computers, it may be impossible to incorporate a large keyboard. When driving, safety is compromised by any visual distraction, and hands are required for controlling the vehicle. The ultimate speech-only device, the telephone, is far more widespread than the PC. Certain manual tasks may also require full visual attention to the focus of the work. Finally, spoken language interfaces offer obvious benefits for individuals challenged with a variety of physical disabilities, such as loss of sight or limitations in physical motion and motor skills. Chapter 18 contains a detailed discussion on spoken language applications.

1.1.2. Speech-to-Speech Translation

Speech-to-speech translation has been depicted for decades in science fiction stories. Imagine questioning a Chinese-speaking conversational partner by speaking English into an unobtrusive device, and hearing real-time replies you can understand. This scenario, like the spoken language interface, requires both speech recognition and speech synthesis technology. In addition, sophisticated multilingual spoken language understanding is needed. This highlights the need for tightly coupled advances in speech recognition, synthesis, and understanding systems, a point emphasized throughout this book.

1.1.3. Knowledge Partners

The ability of computers to process spoken language as proficient as humans will be a landmark to signal the arrival of truly intelligent machines. Alan Turing [29] introduced his famous *Turing test*. He suggested a game, in which a computer's use of language would form the criterion for intelligence. If the machine could win the game, it would be judged intelligent. In Turing's game, you play the role of an interrogator. By asking a series of questions via a teletype, you must determine the identity of the other two participants: a machine and a person. The task of the machine is to fool you into believing it is a person by responding as a person to your questions. The task of the other person is to convince you the other participant is the machine. The critical issue for Turing was that using language as humans do is sufficient as an operational test for intelligence.

The ultimate use of spoken language is to pass the Turing test in allowing future extremely intelligent systems to interact with human beings as knowledge partners in all aspects of life. This has been a staple of science fiction, but its day will come. Such systems require reasoning capabilities and extensive world knowledge embedded in sophisticated search, communication, and inference tools that are beyond the scope of this book. We expect that spoken language technologies described in this book will form the essential enabling mechanism to pass the Turing test.

1.2. SPOKEN LANGUAGE SYSTEM ARCHITECTURE

Spoken language processing refers to technologies related to speech recognition, text-to-speech, and spoken language understanding. A spoken language system has at least one of the following three subsystems: a speech recognition system that converts speech into words, a text-to-speech system that conveys spoken information, and a spoken language understanding system that maps words into actions and that plans system-initiated actions.

There is considerable overlap in the fundamental technologies for these three subareas. Manually created rules have been developed for spoken language systems with limited success. But, in recent decades, data-driven statistical approaches have achieved encouraging results, which are usually based on modeling the speech signal using well-defined statistical algorithms that can automatically extract knowledge from the data. The data-driven approach can be viewed fundamentally as a pattern recognition problem. In fact, speech recognition, text-to-speech conversion, and spoken language understanding can all be regarded as pattern recognition problems. The patterns are either recognized during the runtime operation of the system or identified during system construction to form the basis of runtime generative models such as prosodic templates needed for text-to-speech synthesis. While we use and advocate the statistical approach, we by no means exclude the knowledge engineering approach from consideration. If we have a good set of rules in a given problem area, there is no need to use the statistical approach at all. The problem is that, at time of this writing, we do not have enough knowledge to produce a complete set of high-quality rules. As scientific and theoretical generalizations are made from data collected to construct data-driven systems, better rules may be constructed. Therefore, the rule-based and statistical approaches are best viewed as complementary.

1.2.1. Automatic Speech Recognition

A source-channel mathematical model described in Chapter 3 is often used to formulate speech recognition problems. As illustrated in Figure 1.1, the speaker's mind decides the source word sequence $\mathbf{W}$ that is delivered through his/her text generator. The source is passed through a noisy communication channel that consists of the speaker's vocal apparatus to produce the speech waveform and the speech signal processing component of the speech recognizer. Finally, the speech decoder aims to decode the acoustic signal $\mathbf{X}$ into a word sequence $\hat{\mathbf{W}}$, which is hopefully close to the original word sequence $\mathbf{W}$.

A typical practical speech recognition system consists of basic components shown in the dotted box of Figure 1.2. Applications interface with the decoder to get recognition results that may be used to adapt other components in the system. *Acoustic models* include the representation of knowledge about acoustics, phonetics, microphone and environment variability, gender and dialect differences among speakers, etc. *Language models* refer to a system's knowledge of what constitutes a possible word, what words are likely to co-occur, and in what sequence. The semantics and functions related to an operation a user may wish to perform may also be necessary for the language model. Many uncertainties exist in these areas, associated with speaker characteristics, speech style and rate, recognition of basic

speech segments, possible words, likely words, unknown words, grammatical variation, noise interference, nonnative accents, and confidence scoring of results. A successful speech recognition system must contend with all of these uncertainties. But that is only the beginning. The acoustic uncertainties of the different accents and speaking styles of individual speakers are compounded by the lexical and grammatical complexity and variations of spoken language, which are all represented in the language model.

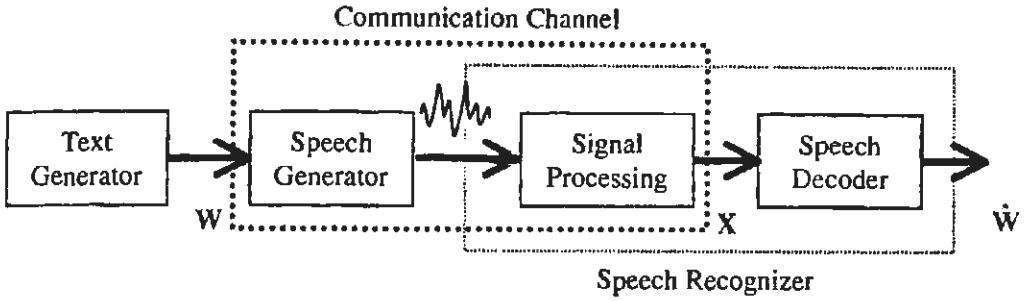


Figure 1.1 A source-channel model for a speech recognition system [15].

The speech signal is processed in the signal processing module that extracts salient feature vectors for the decoder. The decoder uses both acoustic and language models to generate the word sequence that has the maximum posterior probability for the input feature vectors. It can also provide information needed for the adaptation component to modify either the acoustic or language models so that improved performance can be obtained.

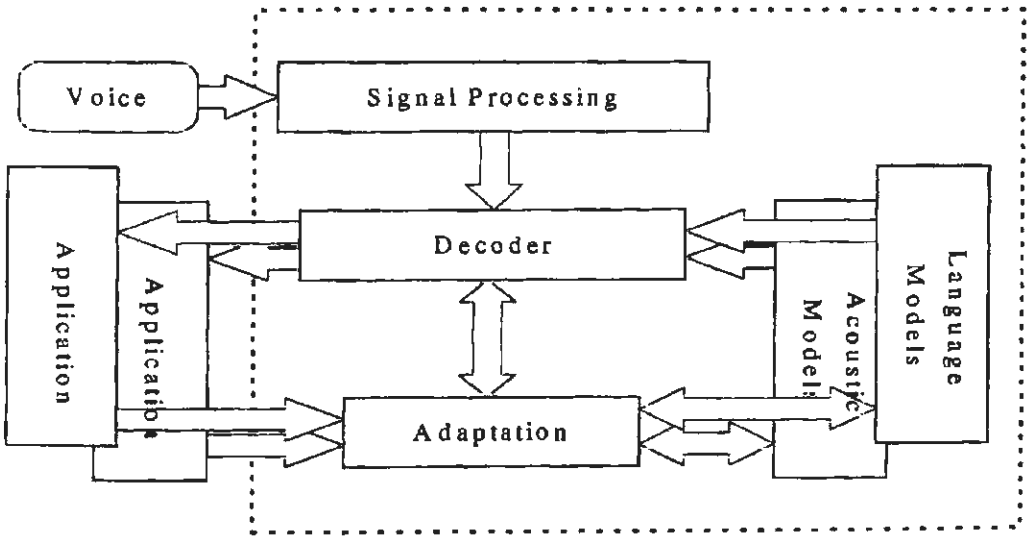


Figure 1.2 Basic system architecture of a speech recognition system [12].

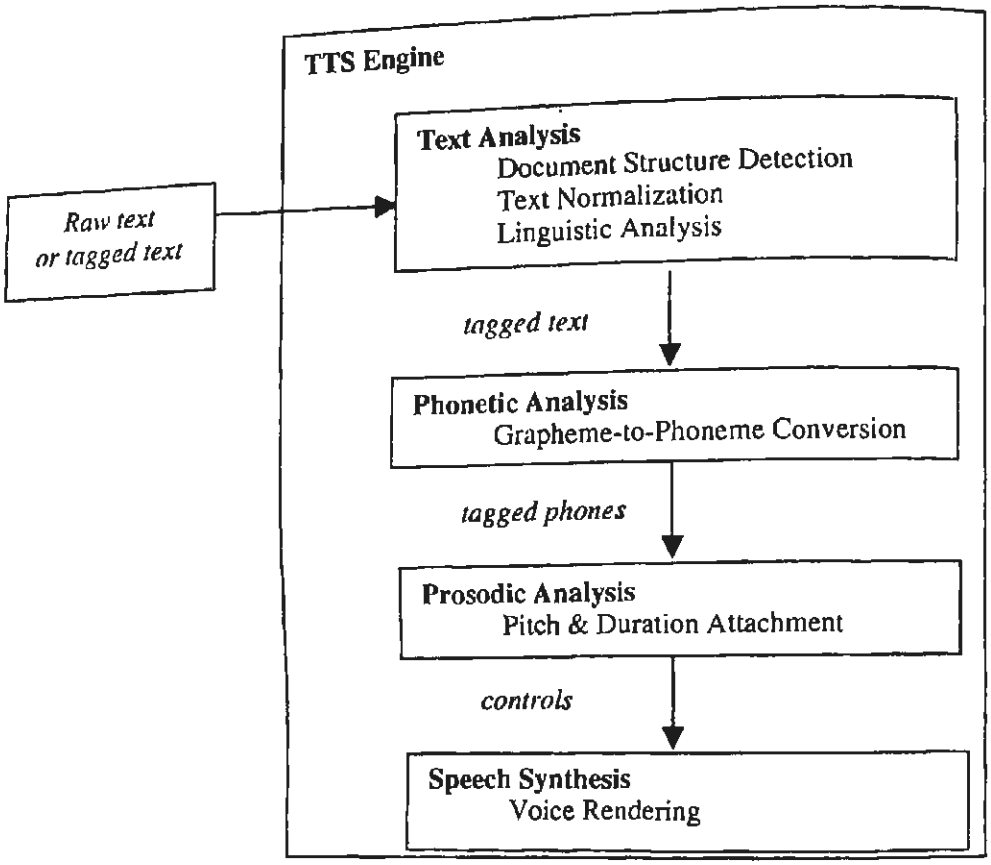


Figure 1.3 Basic system architecture of a TTS system.

1.2.2. Text-to-Speech Conversion

The term *text-to-speech*, often abbreviated as TTS, is easily understood. The task of a text-to-speech system can be viewed as speech recognition in reverse – a process of building a machinery system that can generate human-like speech from any text input to mimic human speakers. TTS is sometimes called *speech synthesis*, particularly in the engineering community.

The conversion of words in written form into speech is nontrivial. Even if we can store a huge dictionary for most common words in English; the TTS system still needs to deal with millions of names and acronyms. Moreover, in order to sound natural, the intonation of the sentences must be appropriately generated.

The development of TTS synthesis can be traced back to the 1930s when Dudley's *Voder*, developed by Bell Laboratories, was demonstrated at the World's Fair [18]. Taking advantage of increasing computation power and storage technology, TTS researchers have been able to generate high-quality commercial multilingual text-to-speech systems, although the quality is inferior to human speech for general-purpose applications.

The basic TTS components are shown in Figure 1.3. The text analysis component normalizes the text to the appropriate form so that it becomes speakable. The input can be

either raw text or tagged. These tags can be used to assist text, phonetic, and prosodic analysis. The phonetic analysis component converts the processed text into the corresponding phonetic sequence, which is followed by prosodic analysis to attach appropriate pitch and duration information to the phonetic sequence. Finally, the speech synthesis component takes the parameters from the fully tagged phonetic sequence to generate the corresponding speech waveform.

Various applications have different degrees of knowledge about the structure and content of the text that they wish to speak so some of the basic components shown in Figure 1.3 can be skipped. For example, some applications may have certain broad requirements such as rate and pitch. These requirements can be indicated with simple command tags appropriately located in the text. Many TTS systems provide a set of markups (tags), so the text producer can better express their semantic intention. An application may know a lot about the structure and content of the text to be spoken to greatly improve speech output quality. For engines providing such support, the *text analysis* phase can be skipped, in whole or in part. If the system developer knows the phonetic form, the phonetic analysis module can be skipped as well. The prosodic analysis module assigns a numeric duration to every phonetic symbol and calculates an appropriate pitch contour for the utterance or paragraph. In some cases, an application may have prosodic contours precalculated by some other process. This situation might arise when TTS is being used primarily for compression, or the prosody is *transplanted* from a real speaker's utterance. In these cases, the quantitative prosodic controls can be treated as special tagged field and sent directly along with the phonetic stream to speech synthesis for voice rendition.

1.2.3. Spoken Language Understanding

Whether a speaker is inquiring about flights to Seattle, reserving a table at a Pittsburgh restaurant, dictating an article in Chinese, or making a stock trade, a spoken language understanding system is needed to interpret utterances in context and carry out appropriate actions. Lexical, syntactic, and semantic knowledge must be applied in a manner that permits cooperative interaction among the various levels of acoustic, phonetic, linguistic, and application knowledge in minimizing uncertainty. Knowledge of the characteristic vocabulary, typical syntactic patterns, and possible actions in any given application context for both interpretation of user utterances and planning system activity are the heart and soul of any spoken language understanding system.

A schematic of a typical spoken language understanding system is shown in Figure 1.4. Such a system typically has a speech recognizer and a speech synthesizer for basic speech input and output, and a *sentence interpretation* component to parse the speech recognition results into semantic forms, which often need *discourse analysis* to track context and resolve ambiguities. The *Dialog Manager* is the central component that communicates with applications and the spoken language understanding modules such as discourse analysis, sentence interpretation, and response generation.

While most components of the system may be partly or wholly generic, the dialog manager controls the flow of conversation tied to the action. The dialog manager is respon-

sible for providing status needed for formulating responses, and maintaining the system's idea of the state of the discourse. The discourse state records the current transaction, dialog goals that motivated the current transaction, current objects in focus (temporary center of attention), the object history list for resolving dependent references, and other status information. The discourse information is crucial for sentence interpretation to interpret utterances in context. Various systems may alter the flow of information implied in Figure 1.4. For example, the dialog manager may be able to supply contextual discourse information or pragmatic inferences, as feedback to guide the recognizer's evaluation of hypotheses at the earliest level of search.

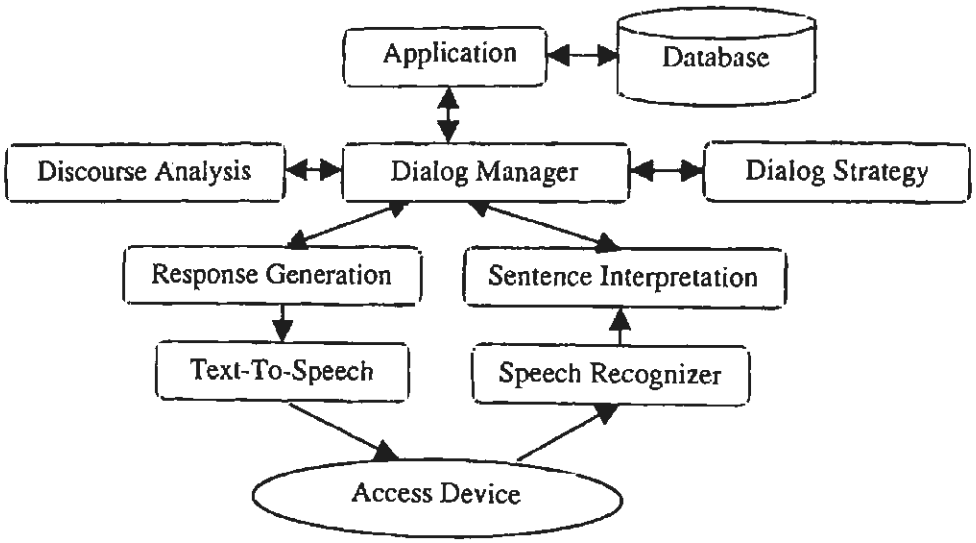


Figure 1.4 Basic system architecture of a spoken language understanding system.

1.3. BOOK ORGANIZATION

We attempt to present a comprehensive introduction to spoken language processing, which includes not only fundamentals but also a practical guide to build a working system that requires knowledge in speech signal processing, recognition, text-to-speech, spoken language understanding, and application integration. Since there is considerable overlap in the fundamental spoken language processing technologies, we have devoted Part I to the foundations needed. Part I contains background on speech production and perception, probability and information theory, and pattern recognition. Parts II, III, IV, and V include chapters on speech processing, speech recognition, speech synthesis, and spoken language systems, respectively. A reader with sufficient background can skip Part I, referring back to it later as needed. For example, the discussion of speech recognition in Part III relies on the pattern recognition algorithms presented in Part I. Algorithms that are used in several chapters

within Part III are also included in Parts I and II. Since the field is still evolving, at the end of each chapter we provide a historical perspective and list further readings to facilitate future research.

1.3.1. Part I: Fundamental Theory

Chapters 2 to 4 provide you with a basic theoretic foundation to better understand techniques that are widely used in modern spoken language systems. These theories include the essence of linguistics, phonetics, probability theory, information theory, and pattern recognition. These chapters prepare you fully to understand the rest of the book.

Chapter 2 discusses the basic structure of spoken language including speech science, phonetics, and linguistics. Chapter 3 covers probability theory and information theory, which form the foundation of modern pattern recognition. Many important algorithms and principles in pattern recognition and speech coding are derived based on these theories. Chapter 4 introduces basic pattern recognition, including decision theory, estimation theory, and a number of algorithms widely used in speech recognition. Pattern recognition forms the core of most of the algorithms used in spoken language processing.

1.3.2. Part II: Speech Processing

Part II provides you with necessary speech signal processing knowledge that is critical to spoken language processing. Most of what discuss here is traditionally the subject of electrical engineering.

Chapters 5 and 6 focus on how to extract useful information from the speech signal. The basic principles of digital signal processing are reviewed and a number of useful representations for the speech signal are discussed. Chapter 7 covers how to compress these representations for efficient transmission and storage.

1.3.3. Part III: Speech Recognition

Chapters 8 to 13 provide you with an in-depth look at modern speech recognition systems. We highlight techniques that have been proven to work well in real systems and explain in detail how and why these techniques work from both theoretic and practical perspectives.

Chapter 8 introduces hidden Markov models, the most prominent technique used in modern speech recognition systems. Chapters 9 and 11 deal with acoustic modeling and language modeling respectively. Because environment robustness is critical to the success of practical systems, we devote Chapter 10 to discussing how to make systems less affected by environment noises. Chapters 12 and 13 deal in detail with how to efficiently implement the decoder for speech recognition. Chapter 12 discusses a number of basic search algorithms, and Chapter 13 covers large vocabulary speech recognition. Throughout our discussion, Microsoft's Whisper speech recognizer is used as a case study to illustrate the methods introduced in these chapters.

1.3.4. Part IV: Text-to-Speech Systems

In Chapters 14 through 16, we discuss proven techniques in building text-to-speech systems. The synthesis system consists of major components found in speech recognition systems, except that they are in the reverse order.

Chapter 14 covers the analysis of written documents and the text needed to support spoken rendition, including the interpretation of audio markup commands, interpretation of numbers and other symbols, and conversion from orthographic to phonetic symbols. Chapter 15 focuses on the generation of pitch and duration controls for linguistic and emotional effect. Chapter 16 discusses the implementation of the synthetic voice, and presents algorithms to manipulate a limited voice data set to support a wide variety of pitch and duration controls required by the text analysis. We highlight the importance of trainable synthesis, with Microsoft's Whistler TTS system as an example.

1.3.5. Part V: Spoken Language Systems

As discussed in Section 1.1, spoken language applications motivate spoken language R&D. The central component is the spoken language understanding system. Since it is closely related to applications, we group it together with application and interface design.

Chapter 17 covers spoken language understanding. The output of the recognizer requires interpretation and action in a particular application context. This chapter details useful strategies for dialog management, and the coordination of all the speech and system resources to accomplish a task for a user. Chapter 18 concludes the book with a discussion of important principles for building spoken language interfaces and applications, including general human interface design goals, and interaction with other modalities in specific application contexts. Microsoft's MiPad is used as a case study to illustrate a number of issues in developing spoken language and multimodal applications.

1.4. TARGET AUDIENCES

This book can serve a variety of audiences:

Integration engineers: Software engineers who want to build spoken language systems, but who do not want to learn detailed speech technology internals, will find plentiful relevant material, including application design and software interfaces. Anyone with a professional interest in aspects of speech applications, integration, and interfaces can also achieve enough understanding of how the core technologies work, to allow them to take full advantage of state-of-the-art capabilities.

Speech technology engineers: Engineers and researchers working on various subspecialties within the speech field will find this book a useful guide to understanding related technologies in sufficient depth to help them gain insight on where their own approaches overlap with, or diverge from, their neighbors' common practice.

Graduate students: This book can serve as a primary textbook in a graduate or advanced undergraduate speech analysis or language engineering course. It can serve as a sup-

plementary textbook in some applied linguistics, digital signal processing, computer science, artificial intelligence, and possibly psycholinguistics course.

Linguists: As the practice of linguistics increasingly shifts to empirical analysis of real-world data, students and professional practitioners alike should find a comprehensive introduction to the technical foundations of computer processing of spoken language helpful. The book can be read at different levels and through different paths, for readers with differing technical skills and background knowledge.

Speech scientists: Researchers engaged in professional work on issues related to normal or pathological speech may find this complete exposition of the state-of-the-art in computer modeling of generation and perception of speech interesting.

Business planners: Increasingly, business and management functions require some level of insight into the vocabulary and common practices of technology development. While not the primary audience, managers, marketers, and others with planning responsibilities and sufficient technical background will find portions of this book useful in evaluating competing proposals, and in making business decisions related to the speech technology components.

1.5. HISTORICAL PERSPECTIVE AND FURTHER READING

Spoken language processing is a diverse field that relies on knowledge of language at the levels of signal processing, acoustics, phonology, phonetics, syntax, semantics, pragmatics, and discourse. The foundations of spoken language processing lie in computer science, electrical engineering, linguistics, and psychology. In the 1970s an ambitious speech understanding project was funded by DARPA, which led to many seminal systems and technologies [17]. A number of human language technology projects funded by DARPA in the 1980s and 1990s further accelerated the progress, as evidenced by many papers published in *The Proceedings of the DARPA Speech and Natural Language/Human Language Workshop*. The field is still rapidly progressing and there are a number of excellent review articles and introductory books. We provide a brief list here. More detailed references can be found within each chapter of this book. Gold and Morgan's *Speech and Audio Signal Processing* [10] also has a strong historical perspective on spoken language processing.

Hyde [14] and Reddy [24] provided an excellent review of early speech recognition work in the 1970s. Some of the principles are still applicable to today's speech recognition research. Waibel and Lee assembled many seminal papers in *Readings in Speech Recognition Speech Recognition* [31]. There are a number of excellent books on modern speech recognition [1, 13, 15, 22, 23].

Where does the state of the art speech recognition system stand today? A number of different recognition tasks can be used to compare the recognition error rate of people vs. machines. Table 1.1 shows five typical recognition tasks with vocabularies ranging from 10 to 5000 words speaker-independent continuous speech recognition. The Wall Street Journal Dictation (WSJ) Task has a 5000-word vocabulary as a continuous dictation application for the WSJ articles. In Table 1.1, the error rate for machines is based on state of the art speech

recognizers such as systems described in Chapter 9, and the error rate of humans is based on a range of subjects tested on the similar task. We can see the error rate of humans is at least 5 times smaller than machines except for the sentences that are generated from a trigram language model, where the sentences have the perfect match between humans and machines so humans cannot use high-level knowledge that is not used in machines.[1]

Table 1.1 Word error rate comparisons between human and machines on similar tasks.

Tasks	Vocabulary	Humans	Machines
Connected digits	10	0.009%	0.72%
Alphabet letters	26	1%	5%
Spontaneous telephone speech	2000	3.8%	36.7%
WSJ with clean speech	5000	0.9%	4.5%
WSJ with noisy speech (10-db SNR)	5000	1.1%	8.6%
Clean speech based on trigram sentences	20,000	7.6%	4.4%

We can see that humans are far more robust than machines for normal tasks. The error rate for machine spontaneous conversational telephone speech recognition is above 35%, more than a factor 10 higher than humans on the similar task. In addition, the error rate of humans does not increase as dramatically as machines when the environment becomes noisy (from quiet to 10-db SNR environments on the WSJ task). The relative error rate of humans increases from 0.9% to 1.1% (1.2 times), while the error rate of CSR systems increases from 4.5% to 8.6% (1.9 times). One interesting experiment is that when we generated sentences using the WSJ trigram language model (cf. Chapter 11), the difference between humans and machines disappears (the last row in Table 1.1). In fact, the error rate of humans is even higher than machines. This is because both humans and machines have the same high-level syntactic and semantic models. The test sentences are somewhat random to humans but perfect to machines that used the same trigram model for decoding. This experiment indicates humans make more effective use of semantic and syntactic constraints for improved speech recognition in meaningful conversation. In addition, machines don't have attention problems as humans do on random sentences.

Fant [7] gave an excellent introduction to speech production. Early reviews of text-to-speech synthesis can be found in [3, 8, 9]. Sagisaka [26] and Carlson [6] provide more recent reviews of progress in speech synthesis. A more detailed treatment can be found in [19, 30].

Where does the state of the art text to speech system stand today? Unfortunately, like speech recognition, this is not a solved problem either. Although machine storage capabilities are improving, the quality remains a challenge for many researchers if we want to pass the Turing test.

[1] Some of these experiments were conducted at Microsoft with only a small number of human subjects (3-5 people), which is not statistically significant. Nevertheless, the experiments give some interesting insight on the performance of humans and machines.

Spoken language understanding is deeply rooted in speech recognition research. There are a number of good books on spoken language understanding [2, 5, 16]. Manning and Schutze [20] focuses on statistical methods for language understanding. Like Waibel and Lee, Grosz et al. assembled many foundational papers in *Readings in Natural Language Processing* [11]. More recent reviews of progress in spoken language understanding can be found in [25, 28]. Related spoken language interface design issues can be found in [4, 21, 27, 32].

In comparison to speech recognition and text to speech, spoken language understanding is further away from approaching the level of humans, especially for general-purpose spoken language applications.

A number of good conference proceedings and journals report the latest progress in the field. Major results on spoken language processing are presented at the *International Conference on Acoustics, Speech and Signal Processing (ICASSP), International Conference on Spoken Language Processing (ICSLP), Eurospeech Conference,* the *DARPA Speech and Human Language Technology Workshops,* and many workshops organized by the *European Speech Communications Associations (ESCA)* and *IEEE Signal Processing Society.* Journals include *IEEE Transactions on Speech and Audio Processing, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Computer Speech and Language, Speech Communication,* and *Journal of Acoustical Society of America (JASA).* Research results can also be found at computational linguistics conferences such as the *Association for Computational Linguistics (ACL), International Conference on Computational Linguistics (COLING),* and *Applied Natural Language Processing (ANLP).* The journals *Computational Linguistics* and *Natural Language Engineering* cover both theoretical and practical applications of language research. *Speech Recognition Update* published by TMA Associates is an excellent industry newsletter on spoken language applications.

REFERENCES

[1] Acero, A., *Acoustical and Environmental Robustness in Automatic Speech Recognition*, 1993, Boston, MA, Kluwer Academic Publishers.

[2] Allen, J., *Natural Language Understanding*, 2nd ed., 1995, Menlo Park, CA, The Benjamin/Cummings Publishing Company.

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

[4] Balentine, B., and D. Morgan, *How to Build a Speech Recognition Application*, 1999, Enterprise Integration Group.

[5] Bernsen, N., H. Dybkjar, and L. Dybkjar, *Designing Interactive Speech Systems*, 1998, Springer.

[6] Carlson, R., "Models of Speech Synthesis" in *Voice Communications Between Humans and Machines. National Academy of Sciences*, D.B. Roe and J.G. Wilpon, eds., 1994, Washington, D.C., National Academy of Sciences.

[7] Fant, G., *Acoustic Theory of Speech Production*, 1970, The Hague, NL, Mouton.

[8] Flanagan, J., *Speech Analysis Synthesis and Perception*, 1972, New York, Springer-Verlag.

[9] Flanagan, J., "Voices Of Men And Machines," *Journal of Acoustical Society of America*, 1972, **51**, p. 1375.

[10] Gold, B. and N. Morgan, *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*, 2000, John Wiley and Sons.

[11] Grosz, B., F.S. Jones, and B.L. Webber, *Readings in Natural Language Processing*, 1986, Los Altos, CA, Morgan Kaufmann.

[12] Huang, X., *et al.*, "From Sphinx-II to Whisper -- Make Speech Recognition Usable" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, Kluwer Academic Publishers.

[13] Huang, X.D., Y. Ariki, and M.A. Jack, *Hidden Markov Models for Speech Recognition*, 1990, Edinburgh, U.K., Edinburgh University Press.

[14] Hyde, S.R., "Automatic Speech Recognition: Literature, Survey, and Discussion" in *Human Communication, A Unified Approach*, E.E. David and P.B. Denes, eds. 1972, New York, McGraw Hill.

[15] Jelinek, F., *Statistical Methods for Speech Recognition*, Language, Speech, and Communication, 1998, Cambridge, MA, MIT Press.

[16] Jurafsky, D. and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2000, Upper Saddle River, NJ, Prentice Hall.

[17] Klatt, D., "Review of the ARPA Speech Understanding Project," *Journal of Acoustical Society of America*, 1977, **62**(6), pp. 1324-1366.

[18] Klatt, D., "Review of Text-to-Speech Conversion for English," *Journal of Acoustical Society of America*, 1987, **82**, pp. 737-793.

[19] Kleijn, W.B. and K.K. Paliwal, *Speech Coding and Synthesis*, 1995, Amsterdam, Netherlands, Elsevier.

[20] Manning, C. and H. Schutze, *Foundations of Statistical Natural Language Processing*, 1999, MIT Press, Cambridge, USA.

[21] Markowitz, J., *Using Speech Recognition*, 1996, Prentice Hall.

[22] Mori, R.D., *Spoken Dialogues with Computers*, 1998, London, UK, Academic Press.

[23] Rabiner, L.R. and B.H. Juang, *Fundamentals of Speech Recognition*, May, 1993, Prentice-Hall.

[24] Reddy, D.R., "Speech Recognition by Machine: A Review," *IEEE Proc.*, 1976, **64**(4), pp. 502-531.

[25] Sadek, D. and R.D. Mori, "Dialogue Systems" in *Spoken Dialogues with Computers*, R.D. Mori, Editor 1998, London, UK, pp. 523-561, Academic Press.

[26] Sagisaka, Y., "Speech Synthesis from Text," *IEEE Communication Magazine*, 1990(1).

[27] Schmandt, C., *Voice Communication with Computers*, 1994, New York, NY, Van Nostrand Reinhold.

[28] Seneff, S., "The Use of Linguistic Hierarchies in Speech Understanding," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[29] Turing, A.M., "Computing Machinery and Intelligence," *Mind*, 1950, **LIX**(236), pp. 433-460.

[30] van Santen, J., *et al.*, *Progress in Speech Synthesis*, 1997, New York, Springer-Verlag.

[31] Waibel, A.H. and K.F. Lee, *Readings in Speech Recognition*, 1990, San Mateo, CA, Morgan Kaufman Publishers.

[32] Weinschenk, S. and D. Barker, *Designing Effective Speech Interfaces*, 2000, John Wiley & Sons, Inc.

PART I

FUNDAMENTAL THEORY

CHAPTER 2

Spoken Language Structure

Spoken language is used to communicate information from a speaker to a listener. Speech production and perception are both important components of the speech chain. Speech begins with a thought and intent to communicate in the brain, which activates muscular movements to produce speech sounds. A listener receives it in the auditory system, processing it for conversion to neurological signals the brain can understand. The speaker continuously monitors and controls the vocal organs by receiving his or her own speech as feedback.

Considering the universal components of speech communication as shown in Figure 2.1, the fabric of spoken interaction is woven from many distinct elements. The speech production process starts with the semantic message in a person's mind to be transmitted to the listener via speech. The computer counterpart to the process of message formulation is the application semantics that creates the concept to be expressed. After the message is created,

the next step is to convert the message into a sequence of words. Each word consists of a sequence of phonemes that corresponds to the pronunciation of the words. Each sentence also contains a prosodic pattern that denotes the duration of each phoneme, intonation of the sentence, and loudness of the sounds. Once the language system finishes the mapping, the talker executes a series of neuromuscular signals. The neuromuscular commands perform articulatory mapping to control the vocal cords, lips, jaw, tongue, and velum, thereby producing the sound sequence as the final output. The speech understanding process works in reverse order. First the signal is passed to the cochlea in the inner ear, which performs frequency analysis as a filter bank. A neural transduction process follows and converts the spectral signal into activity signals on the auditory nerve, corresponding roughly to a feature extraction component. Currently, it is unclear how neural activity is mapped into the language system and how message comprehension is achieved in the brain.

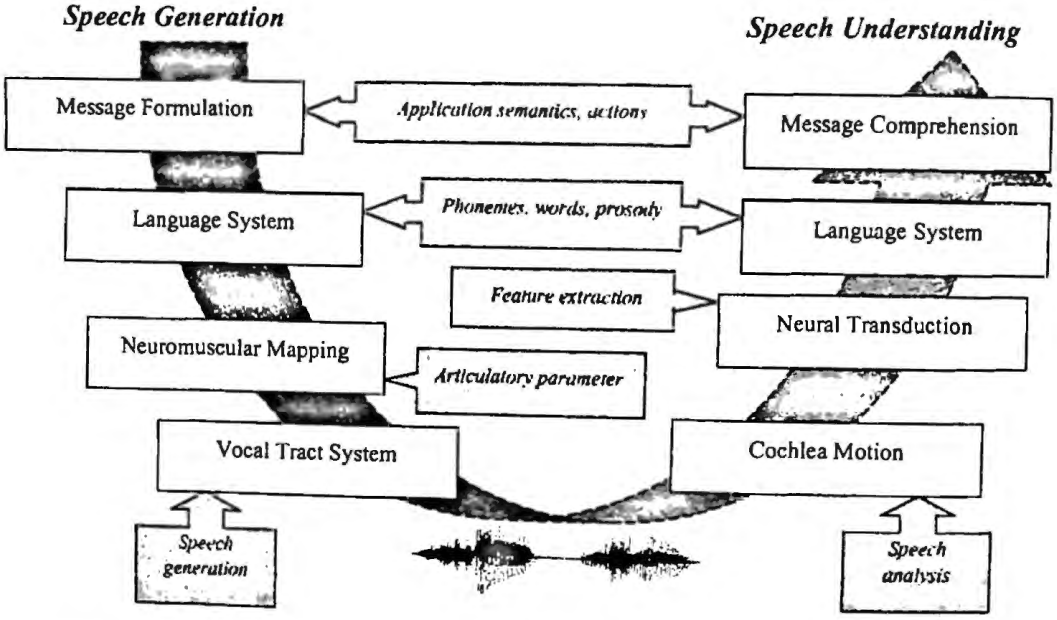


Figure 2.1 The underlying determinants of speech generation and understanding. The gray boxes indicate the corresponding computer system components for spoken language processing.

Speech signals are composed of analog sound patterns that serve as the basis for a discrete, symbolic representation of the spoken language – phonemes, syllables, and words. The production and interpretation of these sounds are governed by the syntax and semantics of the language spoken. In this chapter, we take a bottom up approach to introduce the basic concepts from sound to phonetics and phonology. Syllables and words are followed by syntax and semantics, which form the structure of spoken language processing. The examples in this book are drawn primarily from English, though they are relevant to other languages.

2.1. SOUND AND HUMAN SPEECH SYSTEMS

In this section, we briefly review human speech production and perception systems. We hope spoken language research will enable us to build a computer system that is as good as or better than our own speech production and understanding system.

2.1.1. Sound

Sound is a longitudinal pressure wave formed of compressions and rarefactions of air molecules, in a direction parallel to that of the application of energy. Compressions are zones where air molecules have been forced by the application of energy into a tighter-than-usual configuration, and rarefactions are zones where air molecules are less tightly packed. The alternating configurations of compression and rarefaction of air molecules along the path of an energy source are sometimes described by the graph of a sine wave as shown in Figure 2.2. In this representation, crests of the sine curve correspond to moments of maximal compression and troughs to moments of maximal rarefaction.

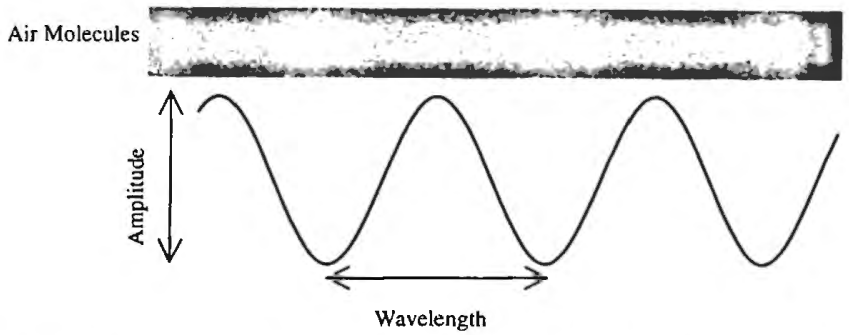


Figure 2.2 Application of sound energy causes alternating compression/rarefaction of air molecules, described by a sine wave. There are two important parameters, amplitude and wavelength, to describe a sine wave. Frequency [cycles/second measured in Hertz (Hz)] is also used to measure of the waveform.

The use of the sine graph in Figure 2.2 is only a notational convenience for charting local pressure variations over time, since sound does not form a transverse wave, and the air particles are just *oscillating in place* along the line of application of energy. The speed of a sound pressure wave in air is approximately $331.5 + 0.6T_c m/s$, where T_c is the Celsius temperature.

The amount of work done to generate the energy that sets the air molecules in motion is reflected in the amount of displacement of the molecules from their resting position. This *degree of displacement* is measured as the amplitude of a sound as shown in Figure 2.2. Because of the wide range, it is convenient to measure sound amplitude on a logarithmic scale in *decibels* (dB). A decibel scale is a means for comparing the intensity of two sounds:

$$10 \log_{10} (I / I_0) \tag{2.1}$$

where I and I_0 are the two intensity levels, with intensity being proportional to the square of the sound pressure P.

Sound pressure level (SPL) is a measure of absolute sound pressure P in dB:

$$SPL(dB) = 20 \log_{10} \left(\frac{P}{P_0} \right) \tag{2.2}$$

where the reference 0 dB corresponds to the threshold of hearing, which is $P_0 = 0.0002 \mu bar$ for a tone of 1kHz. The speech conversation level at 3 feet is about 60 dB SPL, and a jackhammer's level is about 120 dB SPL. Alternatively, watts/meter^2 units are often used to indicate intensity. We can bracket the limits of human hearing as shown in Table 2.1. On the low end, the human ear is quite sensitive. A typical person can detect sound waves having an intensity of 10^{-12} W/m^2 (the *threshold of hearing* or TOH). This intensity corresponds to a pressure wave affecting a given region by only one-billionth of a centimeter of molecular motion. On the other end, the most intense sound that can be safely detected without suffering physical damage is one trillion times more intense than the TOH. 0 dB begins with the TOH and advances logarithmically. The faintest audible sound is arbitrarily assigned a value of 0 dB, and the loudest sounds that the human ear can tolerate are about 120 dB.

Table 2.1 Intensity and decibel levels of various sounds.

Sound	dB Level	Times > TOH
Threshold of hearing (TOH: $10^{-12} W/m^2$)	0	10^0
Light whisper	10	10^1
Quiet living room	20	10^2
Quiet conversation	40	10^4
Average office	50	10^5
Normal conversation	60	10^6
Busy city street	70	10^7
Acoustic guitar – 1 ft. away	80	10^8
Heavy truck traffic	90	10^9
Subway from platform	100	10^{10}
Power tools	110	10^{11}
Pain threshold of ear	120	10^{12}
Airport runway	130	10^{13}
Sonic boom	140	10^{14}
Permanent damage to hearing	150	10^{15}
Jet engine, close up	160	10^{16}
Rocket engine	180	10^{18}
Twelve ft. from artillery cannon muzzle ($10^{10} W/m^2$)	220	10^{22}

The absolute threshold of hearing is the maximum amount of energy of a pure tone that cannot be detected by a listener in a noise free environment. The absolute threshold of hearing is a function of frequency that can be approximated by

$$T_q(f) = 3.64(f/1000)^{-0.8} - 6.5e^{-0.6(f/1000-3.3)^2} + 10^{-3}(f/1000)^4 \quad (dB\ SPL) \tag{2.3}$$

and is plotted in Figure 2.3.

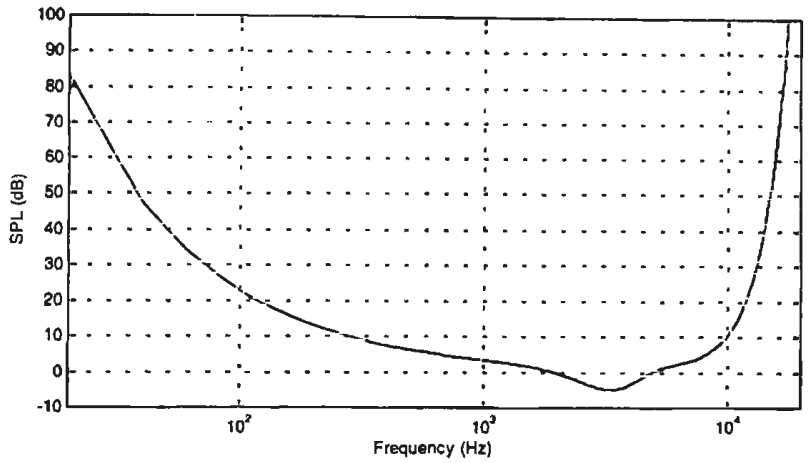


Figure 2.3 The sound pressure level (SPL) level in dB of the absolute threshold of hearing as a function of frequency. Sounds below this level are inaudible. Note that below 100 Hz and above 10 kHz this level rises very rapidly. Frequency goes from 20 Hz to 20 kHz and is plotted in a logarithmic scale from Eq. (2.3).

Let's compute how the pressure level varies with distance for a sound wave emitted by a point source located a distance r away. Assuming no energy absorption or reflection, the sound wave of a point source is propagated in a spherical front, such that the energy is the same for the sphere's surface at all radius r. Since the surface of a sphere of radius r is $4\pi r^2$, the sound's energy is inversely proportional to r^2, so that every time the distance is doubled, the sound pressure level decreases by 6 dB. For the point sound source, the energy (E) transported by a wave is proportional to the square of the amplitude (A) of the wave and the distance (r) between the sound source and the listener:

$$E \propto \frac{A^2}{r^2} \tag{2.4}$$

The typical sound intensity of a speech signal one inch away (close-talking microphone) from the talker is 1 Pascal = 10μbar, which corresponds to 94 dB SPL. The typical sound intensity 10 inches away from a talker is 0.1 Pascal = 1μbar, which corresponds to 74 dB SPL.

2.1.2. Speech Production

We review here basic human speech production systems, which have influenced research on speech coding, synthesis, and recognition.

2.1.2.1. Articulators

Speech is produced by air-pressure waves emanating from the mouth and the nostrils of a speaker. In most of the world's languages, the inventory of *phonemes,* as discussed in Section 2.2.1, can be split into two basic classes:

- consonants – articulated in the presence of constrictions in the throat or obstructions in the mouth (tongue, teeth, lips) as we speak.
- vowels – articulated without major constrictions and obstructions.

The sounds can be further partitioned into subgroups based on certain articulatory properties. These properties derive from the anatomy of a handful of important articulators and the places where they touch the boundaries of the human vocal tract. Additionally, a large number of muscles contribute to articulator positioning and motion. We restrict ourselves to a schematic view of only the major articulators, as diagrammed in Figure 2.4. The

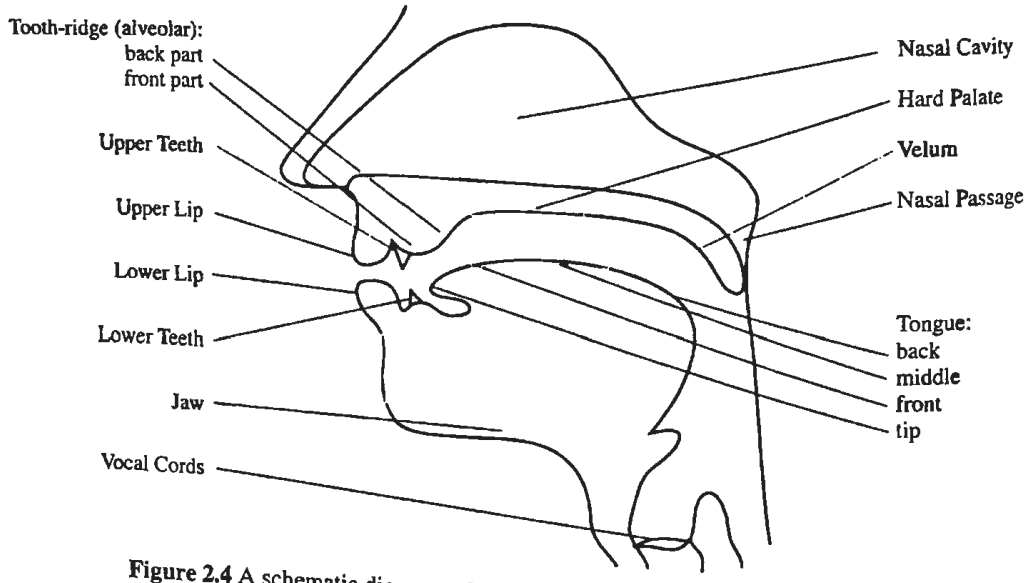


Figure 2.4 A schematic diagram of the human speech production apparatus.

gross components of the speech production apparatus are the lungs, trachea, larynx (organ of voice production), pharyngeal cavity (throat), oral and nasal cavity. The pharyngeal and oral cavities are typically referred to as the vocal tract, and the nasal cavity as the nasal tract. As illustrated in Figure 2.4, the human speech production apparatus consists of:

- *Lungs*: source of air during speech.
- *Vocal cords (larynx):* when the vocal folds are held close together and oscillate against one another during a speech sound, the sound is said to be *voiced.* When the folds are too slack or tense to vibrate periodically, the sound is said to be *unvoiced.* The place where the vocal folds come together is called the *glottis.*
- *Velum (soft palate)*: operates as a *valve*, opening to allow passage of air (and thus resonance) through the nasal cavity. Sounds produced with the flap open include *m* and *n*.
- *Hard palate*: a long relatively hard surface at the roof inside the mouth, which, when the tongue is placed against it, enables consonant articulation.
- *Tongue*: flexible articulator, shaped away from the palate for vowels, placed close to or on the palate or other hard surfaces for consonant articulation.
- *Teeth*: another place of articulation used to brace the tongue for certain consonants.
- *Lips*: can be rounded or spread to affect vowel quality, and closed completely to stop the oral air flow in certain consonants (*p, b, m*).

2.1.2.2. The Voicing Mechanism

The most fundamental distinction between sound types in speech is the voiced/voiceless distinction. Voiced sounds, including vowels, have in their time and frequency structure a roughly regular pattern that voiceless sounds, such as consonants like *s*, lack. Voiced sounds typically have more energy as shown in Figure 2.5. We see here the waveform of the word *sees*, which consists of three phonemes: an unvoiced consonant */s/*, a vowel */iy/*, and a voiced consonant */z/*.

What in the speech production mechanism creates this fundamental distinction? When the vocal folds vibrate during phoneme articulation, the phoneme is considered voiced; otherwise it is unvoiced. Vowels are voiced throughout their duration. The distinct vowel *timbres* are created by using the tongue and lips to shape the main oral resonance cavity in different ways. The vocal folds vibrate at slower or faster rates, from as low as 60 cycles per second (Hz) for a large man, to as high as 300 Hz or higher for a small woman or child. The rate of cycling (opening and closing) of the vocal folds in the larynx during phonation of voiced sounds is called the *fundamental frequency*. This is because it sets the periodic baseline for all higher-frequency harmonics contributed by the pharyngeal and oral resonance

cavities above. The fundamental frequency also contributes more than any other single factor to the perception of *pitch* (the semi-musical rising and falling of voice tones) in speech.

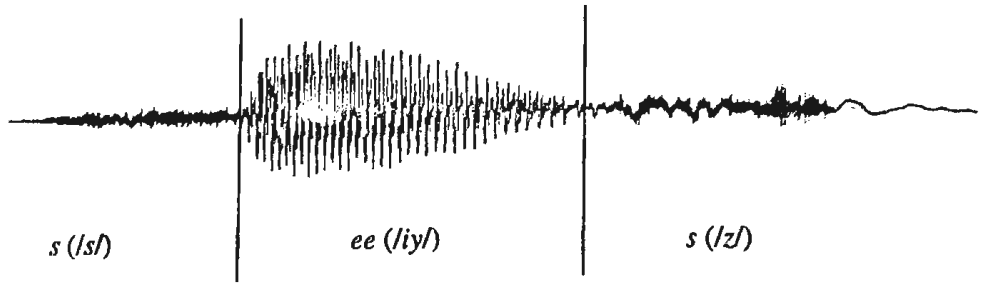


Figure 2.5 Waveform of *sees*, showing a voiceless phoneme */s/*, followed by a voiced sound, the vowel */iy/*. The final sound, */z/*, is a type of voiced consonant.

The glottal cycle is illustrated in Figure 2.6. At stage (a), the vocal folds are closed and the air stream from the lungs is indicated by the arrow. At some point, the air pressure on the underside of the barrier formed by the vocal folds increases until it overcomes the resistance of the vocal fold closure and the higher air pressure below blows them apart (b). However, the tissues and muscles of the larynx and the vocal folds have a natural elasticity which tends to make them fall back into place rapidly, once air pressure is temporarily equalized (c). The successive airbursts resulting from this process are the source of energy for all voiced sounds. The time for a single open-close cycle depends on the stiffness and size of the vocal folds and the amount of subglottal air pressure. These factors can be controlled by a speaker to raise and lower the perceived frequency or pitch of a voiced sound.

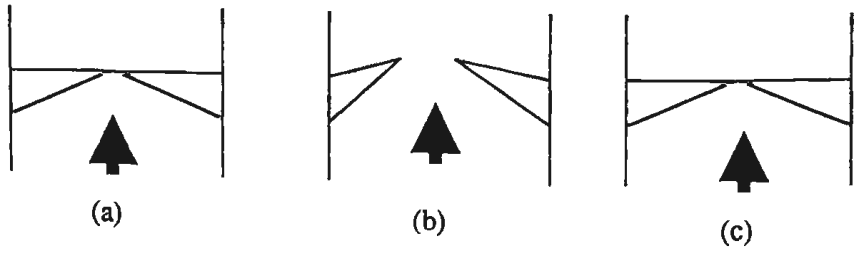


Figure 2.6 Vocal fold cycling at the larynx. (a) Closed with sub-glottal pressure buildup; (b) trans-glottal pressure differential causing folds to blow apart; (c) pressure equalization and tissue elasticity forcing temporary reclosure of vocal folds, ready to begin next cycle.

The waveform of air pressure variations created by this process can be described as a periodic flow, in cubic centimeters per second (after [15]). As shown in Figure 2.7, during the time bracketed as *one cycle*, there is no air flow during the initial closed portion. Then as

the glottis opens (open phase), the volume of air flow becomes greater. After a short peak, the folds begin to resume their original position and the air flow declines until complete closure is attained, beginning the next cycle. A common measure is the number of such cycles per second (Hz), or the fundamental frequency (*F0*). Thus the fundamental frequency for the waveform in Figure 2.7 is about 120 Hz.

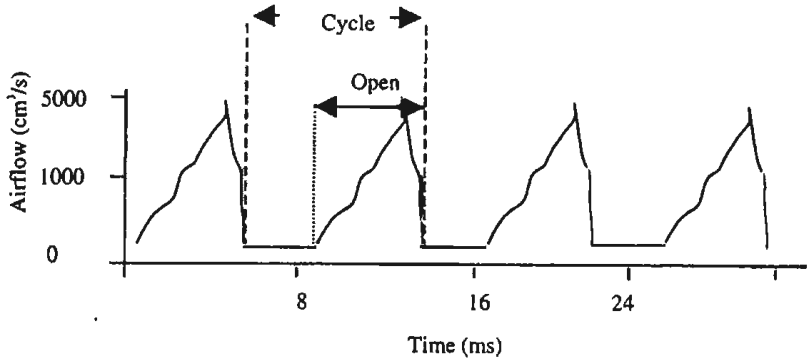


Figure 2.7 Waveform showing air flow during laryngeal cycle.

2.1.2.3. Spectrograms and Formants

Since the glottal wave is periodic, consisting of fundamental frequency (*F0*) and a number of harmonics (integral multiples of *F0*), it can be analyzed as a sum of sine waves as discussed in Chapter 5. The resonances of the vocal tract (above the glottis) are excited by the glottal energy. Suppose, for simplicity, we regard the vocal tract as a straight tube of uniform cross-sectional area, closed at the glottal end, open at the lips. When the shape of the vocal tract changes, the resonances change also. Harmonics near the resonances are emphasized, and, in speech, the resonances of the cavities that are typical of particular articulator configurations (e.g., the different vowel timbres) are called *formants*. The vowels in an actual speech waveform can be viewed from a number of different perspectives, emphasizing either a *cross-sectional* view of the harmonic responses at a single moment, or a longer-term view of the formant track evolution over time. The actual spectral analysis of a vowel at a single time-point, as shown in Figure 2.8, gives an idea of the uneven distribution of energy in resonances for the vowel */iy/* in the waveform for *see*, which is shown in Figure 2.5.

Another view of *sees* of Figure 2.5, called a spectrogram, is displayed in the lower part of Figure 2.9. It shows a long-term frequency analysis, comparable to a complete series of single time-point *cross sections* (such as that in Figure 2.8) ranged alongside one another in time and viewed from *above*.

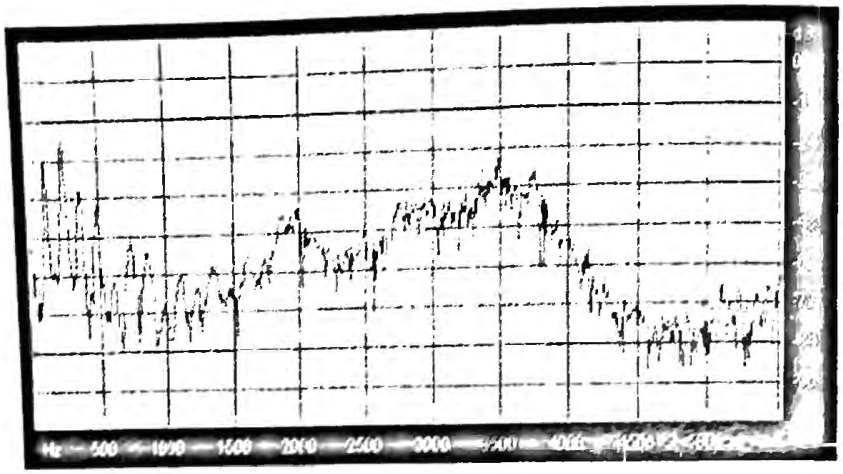


Figure 2.8 A spectral analysis of the vowel */iy/*, showing characteristically uneven distribution of energy at different frequencies.

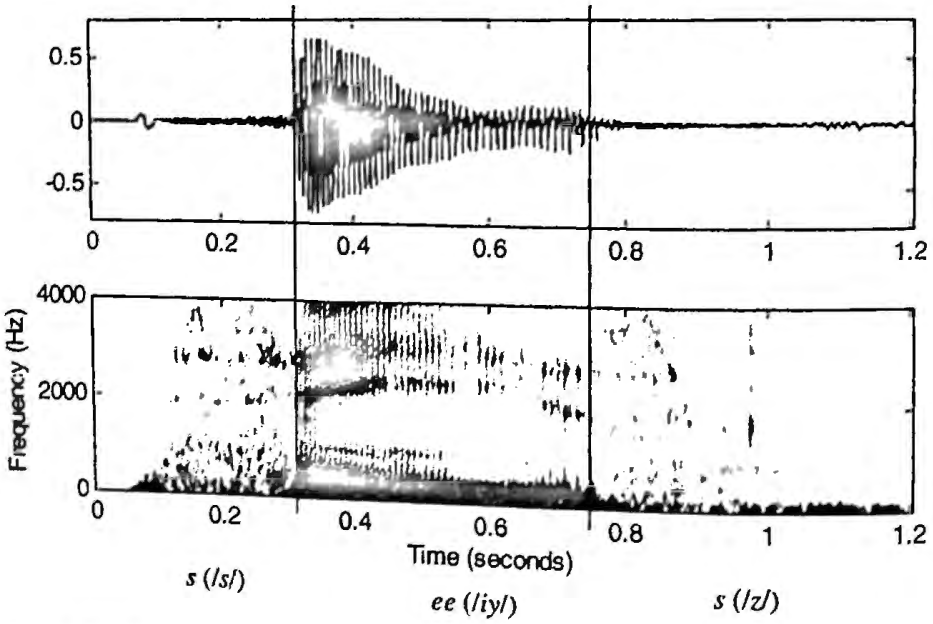


Figure 2.9 The spectrogram representation of the speech waveform *sees* (approximate phone boundaries are indicated with heavy vertical lines).

In the spectrogram of Figure 2.9, the darkness or lightness of a band indicates the relative amplitude or energy present at a given frequency. The dark horizontal bands show the formants, which are harmonics of the fundamental at natural resonances of the vocal tract cavity position for the vowel */iy/* in *see*. The mathematical methods for deriving analyses and representations such as those illustrated above are covered in Chapters 5 and 6.

2.1.3. Speech Perception

There are two major components in the auditory perception system: the peripheral auditory organs (ears) and the auditory nervous system (brain). The ear processes an acoustic pressure signal by first transforming it into a mechanical vibration pattern on the basilar membrane, and then representing the pattern by a series of pulses to be transmitted by the auditory nerve. Perceptual information is extracted at various stages of the auditory nervous system. In this section we focus mainly on the auditory organs.

2.1.3.1. Physiology of the Ear

The human ear, as shown in Figure 2.10, has three sections: the outer ear, the middle ear, and the inner ear. The outer ear consists of the external visible part and the external auditory canal that forms a tube along which sound travels. This tube is about 2.5 cm long and is covered by the eardrum at the far end. When air pressure variations reach the eardrum from the outside, it vibrates, and transmits the vibrations to bones adjacent to its opposite side. The vibration of the eardrum is at the same frequency (alternating compression and rarefaction) as the incoming sound pressure wave. The middle ear is an air-filled space or cavity about 1.3 cm across, and about 6 cm^3 volume. The air travels to the middle ear cavity along the tube (when opened) that connects the cavity with the nose and throat. The oval window shown in Figure 2.10 is a small membrane at the bony interface to the inner ear (cochlea). Since the cochlear walls are bony, the energy is transferred by mechanical action of the stapes into an impression on the membrane stretching over the oval window.

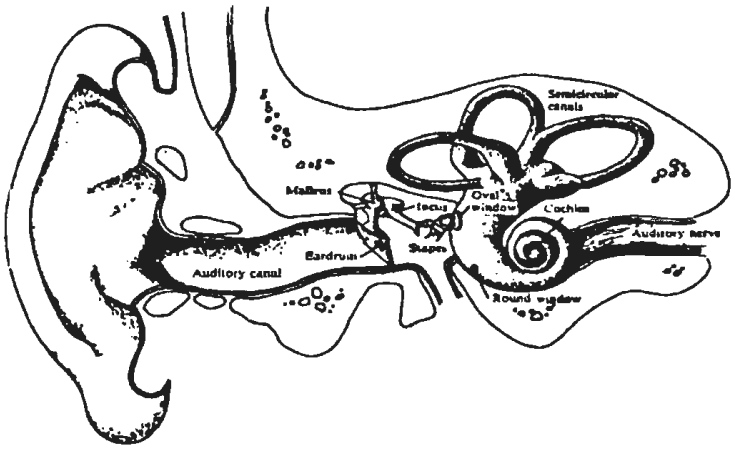


Figure 2.10 The structure of the peripheral auditory system with the outer, middle, and inner ear.

The relevant structure of the inner ear for sound perception is the cochlea, which communicates directly with the auditory nerve, conducting a representation of sound to the brain. The cochlea is a spiral tube about 3.5 cm long, which coils about 2.6 times. The spiral is divided, primarily by the basilar membrane running lengthwise, into two fluid-filled chambers. The cochlea can be roughly regarded as a filter bank, whose outputs are ordered by location, so that a frequency-to-place transformation is accomplished. The filters closest to the cochlear base respond to the higher frequencies, and those closest to its apex respond to the lower.

2.1.3.2. Physical vs. Perceptual Attributes

In psychoacoustics, a basic distinction is made between the perceptual attributes of a sound, especially a speech sound, and the measurable physical properties that characterize it. Each of the perceptual attributes, as listed in Table 2.2, seems to have a strong correlation with one main physical property, but the connection is complex, because other physical properties of the sound may affect perception in complex ways.

Table 2.2 Relation between perceptual and physical attributes of sound.

Physical Quantity	Perceptual Quality
Intensity	Loudness
Fundamental frequency	Pitch
Spectral shape	Timbre
Onset/offset time	Timing
Phase difference in binaural hearing	Location

Although sounds with a greater intensity level usually sound louder, the sensitivity of the ear varies with the frequency and the quality of the sound. One fundamental divergence between physical and perceptual qualities is the phenomenon of non-uniform *equal loudness* perception of tones of varying frequencies. In general, tones of differing pitch have different inherent *perceived loudness*. The sensitivity of the ear varies with the frequency and the quality of the sound. The graph of equal loudness contours adopted by ISO is shown in Figure 2.11. These curves demonstrate the relative insensitivity of the ear to sounds of low frequency at moderate to low intensity levels. Hearing sensitivity reaches a maximum around 4000 Hz, which is near the first resonance frequency of the outer ear canal, and peaks again around 13 kHz, the frequency of the second resonance [38].

Pitch is indeed most closely related to the fundamental frequency. The higher the fundamental frequency, the higher the pitch we perceive. However, discrimination between two pitches depends on the frequency of the lower pitch. Perceived pitch will change as intensity is increased and frequency is kept constant.

In another example of the non-identity of acoustic and perceptual effects, it has been observed experimentally that when the ear is exposed to two or more different tones, it is a common experience that one tone may *mask* the others. Masking is probably best explained

as an upward shift in the hearing threshold of the weaker tone by the louder tone. Pure tones, complex sounds, narrow and broad bands of noise all show differences in their ability to mask other sounds. In general, pure tones close together in frequency mask each other more than tones widely separated in frequency. A pure tone masks tones of higher frequency more effectively than tones of lower frequency. The greater the intensity of the masking tone, the broader the range of the frequencies it can mask [18, 31].

Binaural listening greatly enhances our ability to sense the direction of the sound source. The sense of localization attention is mostly focused on side-to-side discrimination or *lateralization*. Time and intensity cues have different impacts for low frequency and high frequency, respectively. Low-frequency sounds are lateralized mainly on the basis of interaural time difference, whereas high-frequency sounds are localized mainly on the basis of interaural intensity differences [5].

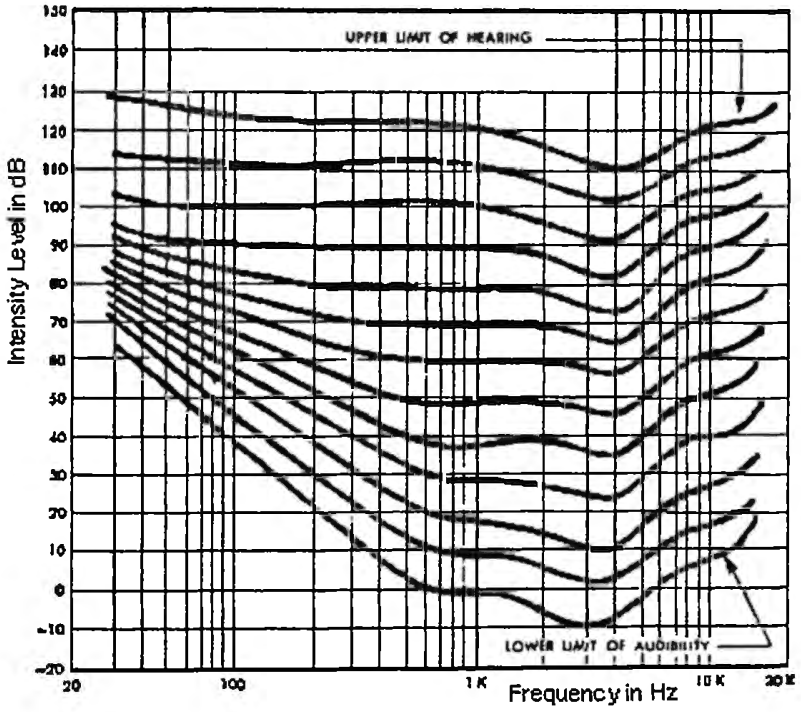


Figure 2.11 Equal-loudness curves indicate that the response of the human hearing mechanism is a function of frequency and loudness levels. This relationship again illustrates the difference between physical dimensions and psychological experience (after ISO 226).

Finally, an interesting perceptual issue is the question of distinctive voice quality. Speech from different people sounds different. Partially this is due to obvious factors, such as differences in characteristic fundamental frequency caused by, for example, the greater mass and length of adult male vocal folds as opposed to female. But there are more subtle

effects as well. In psychoacoustics, the concept of *timbre* (of a sound or instrument) is defined as that attribute of auditory sensation by which a subject can judge that two sounds similarly presented and having the same loudness and pitch are dissimilar. In other words, when all the easily measured differences are controlled, the remaining perception of difference is ascribed to timbre. This is heard most easily in music, where the same note in the same octave played for the same duration on a violin sounds different from a flute. The timbre of a sound depends on many physical variables including a sound's spectral power distribution, its temporal envelope, rate and depth of amplitude or frequency modulation, and the degree of inharmonicity of its harmonics.

2.1.3.3. Frequency Analysis

Researchers have undertaken psychoacoustic experimental work to derive frequency scales that attempt to model the natural response of the human perceptual system, since the cochlea of the inner ear acts as a spectrum analyzer. The complex mechanism of the inner ear and auditory nerve implies that the perceptual attributes of sounds at different frequencies may not be entirely simple or linear in nature. It is well known that the western musical pitch is described in *octaves*[1] and *semi-tones.*[2] The perceived musical pitch of complex tones is basically proportional to the logarithm of frequency. For complex tones, the just noticeable difference for frequency is essentially constant on the octave/semi-tone scale. Musical pitch scales are used in prosodic research (on speech intonation contour generation).

AT&T Bell Labs has contributed many influential discoveries in hearing, such as critical band and articulation index, since the turn of the 20th century [3]. Fletcher's work [14] pointed to the existence of critical bands in the cochlear response. Critical bands are of great importance in understanding many auditory phenomena such as perception of loudness, pitch, and timbre. The auditory system performs frequency analysis of sounds into their component frequencies. The cochlea acts as if it were made up of overlapping filters having bandwidths equal to the critical bandwidth. One class of critical band scales is called *Bark frequency scale*. It is hoped that by treating spectral energy over the Bark scale, a more natural fit with spectral information processing in the ear can be achieved. The Bark scale ranges from 1 to 24 Barks, corresponding to 24 critical bands of hearing as shown in Table 2.3. As shown in Figure 2.12, the perceptual resolution is finer in the lower frequencies. It should be noted that the ear's critical bands are continuous, and a tone of any audible frequency always finds a critical band centered on it. The Bark frequency b can be expressed in terms of the linear frequency (in Hz) by

$$b(f) = 13\arctan(0.00076f) + 3.5 * \arctan\left((f/7500)^2\right) \quad (Bark) \tag{2.5}$$

[1] A tone of frequency f_1 is said to be an octave above a tone with frequency f_2 if and only if $f_1 = 2f_2$.
[2] There are 12 semitones in one octave, so a tone of frequency f_1 is said to be a semitone above a tone with frequency f_2 if and only if $f_1 = 2^{1/12} f_2 = 1.05946 f_2$.

Table 2.3 The Bark frequency scale.

Bark Band #	Edge (Hz)	Center (Hz)
1	100	50
2	200	150
3	300	250
4	400	350
5	510	450
6	630	570
7	770	700
8	920	840
9	1080	1000
10	1270	1170
11	1480	1370
12	1720	1600
13	2000	1850
14	2320	2150
15	2700	2500
16	3150	2900
17	3700	3400
18	4400	4000
19	5300	4800
20	6400	5800
21	7700	7000
22	9500	8500
23	12000	10500
24	15500	13500

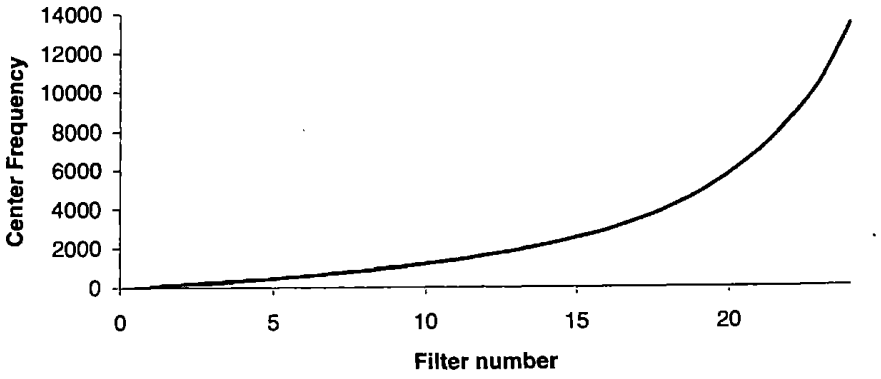


Figure 2.12 The center frequency of 24 Bark frequency filters as illustrated in Table 2.3.

Another such perceptually motivated scale is the mel frequency scale [41], which is linear below 1 kHz, and logarithmic above, with equal numbers of samples taken below and above 1 kHz. The mel scale is based on experiments with simple tones (sinusoids) in which subjects were required to divide given frequency ranges into four perceptually equal intervals or to adjust the frequency of a stimulus tone to be half as high as that of a comparison tone. One mel is defined as one thousandth of the pitch of a 1 kHz tone. As with all such attempts, it is hoped that the mel scale more closely models the sensitivity of the human ear than a purely linear scale and provides for greater discriminatory capability between speech segments. Mel-scale frequency analysis has been widely used in modern speech recognition systems. It can be approximated by:

$$B(f) = 1125 \ln(1 + f/700) \tag{2.6}$$

The mel scale is plotted in Figure 2.13 together with the Bark scale and the bilinear transform (see Chapter 6).

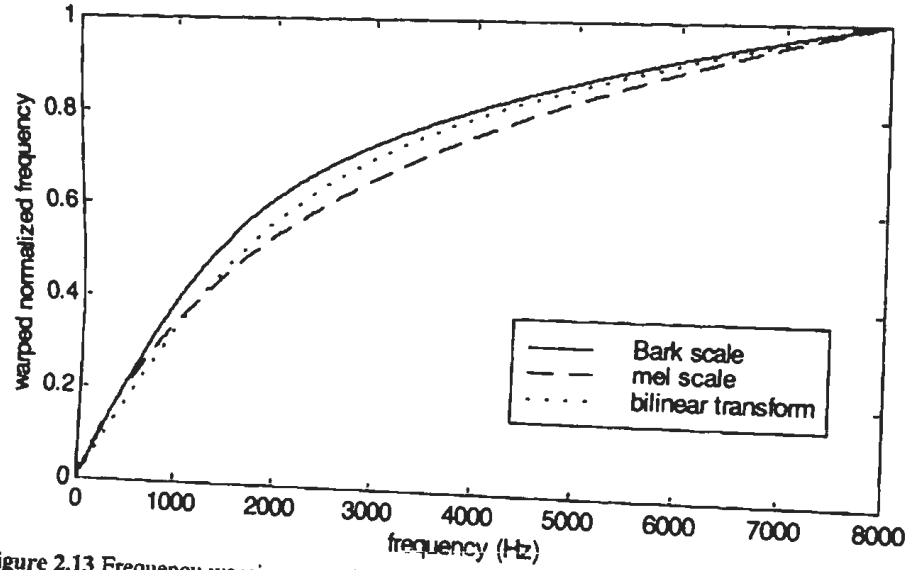


Figure 2.13 Frequency warping according to the Bark scale, ERB scale, mel-scale, and bilinear transform for $\alpha = 0.6$: linear frequency in the x-axis and normalized frequency in the y-axis.

A number of techniques in the modern spoken language system, such as cepstral analysis, and dynamic feature, have benefited tremendously from perceptual research as discussed throughout this book.

2.1.3.4. Masking

Frequency masking is a phenomenon under which one sound cannot be perceived if another sound close in frequency has a high enough level. The first sound *masks* the other one. Fre-

quency-masking levels have been determined empirically, with complicated models that take into account whether the masker is a tone or noise, the masker's level, and other considerations.

We now describe a phenomenon known as *tone-masking noise*. It has been determined empirically that noise with energy E_N (dB) at Bark frequency g masks a tone at Bark frequency b if the tone's energy is below the threshold

$$T_T(b) = E_N - 6.025 - 0.275g + S_m(b-g) \quad (dB\ SPL) \tag{2.7}$$

where the *spread-of-masking* function $S_m(b)$ is given by

$$S_m(b) = 15.81 + 7.5(b+0.474) - 17.5\sqrt{1+(b+0.474)^2} \quad (dB) \tag{2.8}$$

We now describe a phenomenon known as *noise-masking tone*. It has been determined empirically that a tone at Bark frequency g with energy E_T (dB) masks noise at Bark frequency b if the noise energy is below the threshold

$$T_N(b) = E_T - 2.025 - 0.175g + S_m(b-g) \quad (dB\ SPL) \tag{2.9}$$

Masking thresholds are commonly referred to in the literature as Bark scale functions of *just noticeable distortion* (JND). Equation (2.8) can be approximated by a triangular spreading function that has slopes of +25 and –10 dB per Bark, as shown in Figure 2.14.

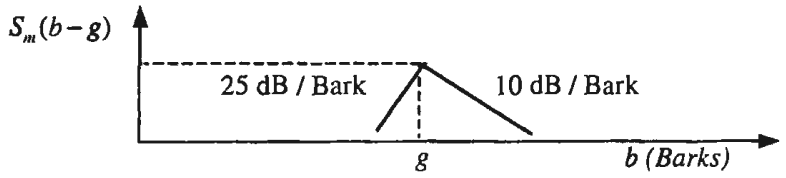


Figure 2.14 Contribution of Bark frequency g to the masked threshold $S_m(b)$.

In Figure 2.15 we show both the threshold of hearing and the masked threshold of a tone at 1 kHz with a 69 dB SPL. The combined masked threshold is the sum of the two in the linear domain

$$T(f) = 10\log_{10}\left(10^{0.1T_h(f)} + 10^{0.1T_T(f)}\right) \tag{2.10}$$

which is approximately the largest of the two.

In addition to frequency masking, there is a phenomenon called temporal masking by which a sound too close in time to another sound cannot be perceived. Whereas premasking tends to last about 5 ms, postmasking can last from 50 to 300 ms. Temporal masking level of a masker with a uniform level starting at 0 ms and lasting 200 ms is shown in Figure 2.16.

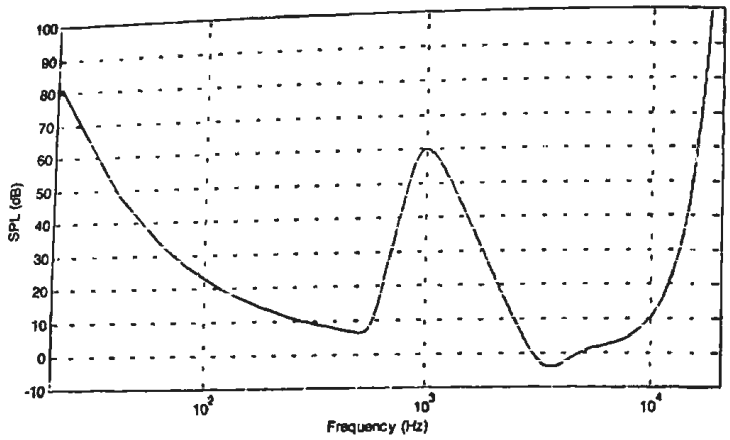


Figure 2.15 Absolute threshold of hearing and spread of masking threshold for a 1 kHz sine-wave masker with a 69 dB SPL. The overall masked threshold is approximately the largest of the two thresholds.

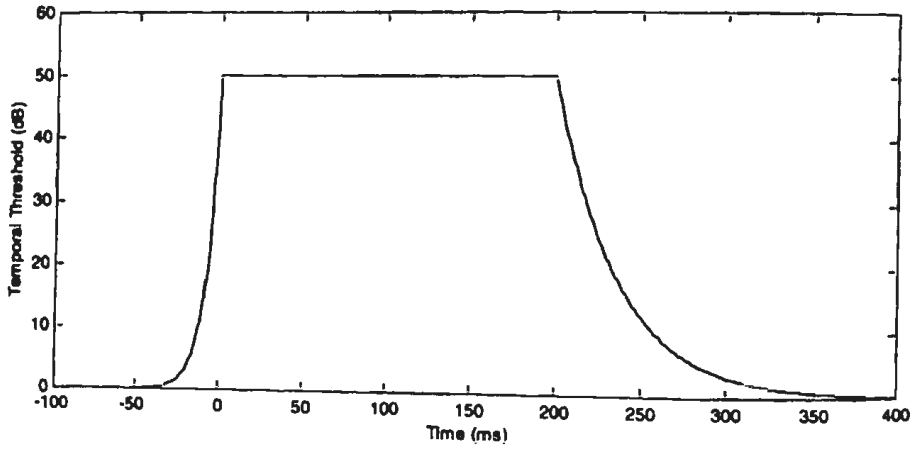


Figure 2.16 Temporal masking level of a masker with a uniform level starting at 0 ms and lasting 200 ms.

2.2. PHONETICS AND PHONOLOGY

We now discuss basic phonetics and phonology needed for spoken language processing. *Phonetics* refers to the study of speech sounds and their production, classification, and transcription. *Phonology* is the study of the distribution and patterning of speech sounds in a language and of the tacit rules governing pronunciation.

2.2.1. Phonemes

Linguist Ferdinand de Saussere (1857-1913) is credited with the observation that the relation between a sign and the object signified by it is arbitrary. The same concept, a certain yellow and black flying social insect, has the sign *honeybee* in English and *mitsubachi* in Japanese.

There is no particular relation between the various pronunciations and the meaning, nor do these pronunciations per se describe the bee's characteristics in any detail. For phonetics, this means that the speech sounds described in this chapter have no inherent meaning, and should be randomly distributed across the lexicon, except as affected by extraneous historical or etymological considerations. The sounds are just a set of arbitrary effects made available by human vocal anatomy. You might wonder about this theory when you observe, for example, the number of words beginning with *sn* that have to do with nasal functions in English: *sneeze*, *snort*, *sniff*, *snot*, *snore*, *snuffle*, etc. But Saussere's observation is generally true, except for obvious onomatopoetic (sound) words like *buzz*.

Like fingerprints, every speaker's vocal anatomy is unique, and this makes for unique vocalizations of speech sounds. Yet language communication is based on commonality of form at the perceptual level. To allow discussion of the commonalities, researchers have identified certain gross characteristics of speech sounds that are adequate for description and classification of words in dictionaries. They have also adopted various systems of notation to represent the subset of phonetic phenomena that are crucial for meaning.

As an analogy, consider the system of computer coding of text characters. In such systems, the *character* is an abstraction, e.g. the Unicode character U+0041. The identifying property of this character is its Unicode name *LATIN CAPITAL LETTER A*. This is a genuine abstraction; no particular realization is necessarily specified. As the Unicode 2.1 standard [1] states:

The Unicode Standard does not define glyph images. The standard defines how characters are interpreted, not how glyphs are rendered. The software or hardware-rendering engine of a computer is responsible for the appearance of the characters on the screen. The Unicode Standard does not specify the size, shape, nor orientation of on-screen characters.

Thus, the U+0041 character can be realized differently for different purposes, and in different sizes with different fonts:

U+0041 → A, *A*, A, A, A, ...

The realizations of the character U+0041 are called glyphs, and there is no distinguished uniquely correct glyph for U+0041. In speech science, the term *phoneme* is used to denote any of the minimal units of speech sound in a language that can serve to distinguish one word from another. We conventionally use the term *phone* to denote a phoneme's acoustic realization. In the example given above, U+0041 corresponds to a phoneme and the various fonts correspond to the phone. For example, English phoneme /t/ have two very different acoustic realizations in the words *sat* and *meter*. You had better treat them as two different phones if you want to build a spoken language system. We will use the terms *phone* or *phoneme* interchangeably to refer to the speaker-independent and context-independent units of meaningful sound contrast. Table 2.4 shows a complete list of phonemes used in American English. The set of phonemes will differ in realization across individual speakers. But phonemes will always function systematically to differentiate meaning in words, just as the phoneme /p/ signals the word *pat* as opposed to the similar-sounding but distinct *bat*. The important contrast distinguishing this pair of words is /p/ vs. /b/.

In this section we concentrate on the basic qualities that *define and differentiate abstract phonemes*. In Section 2.2.1.3 below we consider *why and how phonemes vary* in their actual realizations by different speakers and in different contexts.

Table 2.4 English phonemes used for typical spoken language systems.

Phonemes	Word Examples	Description
iy	*feel, eve, me*	front close unrounded
ih	*fill, hit, lid*	front close unrounded (lax)
ae	*at, carry, gas*	front open unrounded (tense)
aa	*father, ah, car*	back open unrounded
ah	*cut, bud, up*	open-mid back unrounded
ao	*dog, lawn, caught*	open-mid back round
ay	*tie, ice, bite*	diphthong with quality: aa + ih
ax	*ago, comply*	central close mid (schwa)
ey	*ate, day, tape*	front close-mid unrounded (tense)
eh	*pet, berry, ten*	front open-mid unrounded
er	*turn, fur, meter*	central open-mid unrounded rhoti-
ow	*go, own, tone*	back close-mid rounded
aw	*foul, how, our*	diphthong with quality: aa + uh
oy	*toy, coin, oil*	diphthong with quality: ao + ih
uh	*book, pull, good*	back close-mid unrounded (lax)
uw	*tool, crew, moo*	back close round
b	*big, able, tab*	voiced bilabial plosive
p	*put, open, tap*	voiceless bilabial plosive
d	*dig, idea, wad*	voiced alveolar plosive
t	*talk, sat*	voiceless alveolar plosive &
t	*meter*	alveolar flap
g	*gut, angle, tag*	voiced velar plosive
k	*cut, ken, take*	voiceless velar plosive
f	*fork, after, if*	voiceless labiodental fricative
v	*vat, over, have*	voiced labiodental fricative
s	*sit, cast, toss*	voiceless alveolar fricative
z	*zap, lazy, haze*	voiced alveolar fricative
th	*thin, nothing, truth*	voiceless dental fricative
dh	*then, father, scythe*	voiced dental fricative
sh	*she, cushion, wash*	voiceless postalveolar fricative
zh	*genre, azure*	voiced postalveolar fricative
l	*lid*	alveolar lateral approximant
l	*elbow, sail*	velar lateral approximant
r	*red, part, far*	retroflex approximant
y	*yacht, yard*	palatal sonorant glide
w	*with, away*	labiovelar sonorant glide
hh	*help, ahead, hotel*	voiceless glottal fricative
m	*mat, amid, aim*	bilabial nasal
n	*no, end, pan*	alveolar nasal
ng	*sing, anger*	velar nasal
ch	*chin, archer, march*	voiceless alveolar affricate: t + sh
jh	*joy, agile, edge*	voiced alveolar affricate: d + zh

2.2.1.1. Vowels

The tongue shape and positioning in the oral cavity do not form a major constriction of air flow during vowel articulation. However, variations of tongue placement give each vowel its distinct character by changing the resonance, just as different sizes and shapes of bottles give rise to different acoustic effects when struck. The primary energy entering the pharyngeal and oral cavities in vowel production vibrates at the fundamental frequency. The major resonances of the oral and pharyngeal cavities for vowels are called F1 and F2 – the first and second formants, respectively. They are determined by tongue placement and oral tract shape in vowels, and they determine the characteristic timbre or quality of the vowel.

The relationship of F1 and F2 to one another can be used to describe the English vowels. While the shape of the complete vocal tract determines the spectral outcome in a complex, nonlinear fashion, generally F1 corresponds to the back or pharyngeal portion of the cavity, while F2 is determined more by the size and shape of the oral portion, forward of the major tongue extrusion. This makes intuitive sense – the cavity from the glottis to the tongue extrusion is longer than the forward part of the oral cavity, thus we would expect its resonance to be lower. In the vowel of *see*, for example, the tongue extrusion is far forward in the mouth, creating an exceptionally long rear cavity, and correspondingly low F1. The forward part of the oral cavity, at the same time, is extremely short, contributing to higher F2. This accounts for the wide separation of the two lowest dark horizontal bands in Figure 2.9, corresponding to F1 and F2, respectively. Rounding the lips has the effect of extending the front-of-tongue cavity, thus lowering F2. Typical values of F1 and F2 of American English vowels are listed in Table 2.5.

Table 2.5 Phoneme labels and typical formant values for vowels of English.

Vowel Labels	Mean F1 (Hz)	Mean F2 (Hz)
iy (feel)	300	2300
ih (fill)	360	2100
ae (gas)	750	1750
aa (father)	680	1100
ah (cut)	720	1240
ao (dog)	600	900
ax (comply)	720	1240
eh (pet)	570	1970
er (turn)	580	1380
ow (tone)	600	900
uh (good)	380	950
uw (tool)	300	940

The characteristic F1 and F2 values for vowels are sometimes called formant targets, which are ideal locations for perception. Sometimes, due to fast speaking or other limitations on performance, the speaker cannot quite attain an ideal target before the articulators begin shifting to targets for the following phoneme, which is phonetic context dependent. Additionally, there is a special class of vowels that combine two distinct sets of F1/F2 targets.

These are called *diphthongs*. As the articulators move, the initial vowel targets glide smoothly to the final configuration. Since the articulators are working faster in production of a diphthong, sometimes the *ideal* formant target values of the component values are not fully attained. Typical diphthongs of American English are listed in Table 2.6.

Table 2.6 The diphthongs of English.

Diphthong Labels	Components
ay (tie)	*/aa/ → /iy/*
ey (ate)	*/eh/ → /iy/*
oy (coin)	*/ao/ → /iy/*
aw (foul)	*/aa/ → /uw/*

Figure 2.17 shows the first two formants for a number of typical vowels.

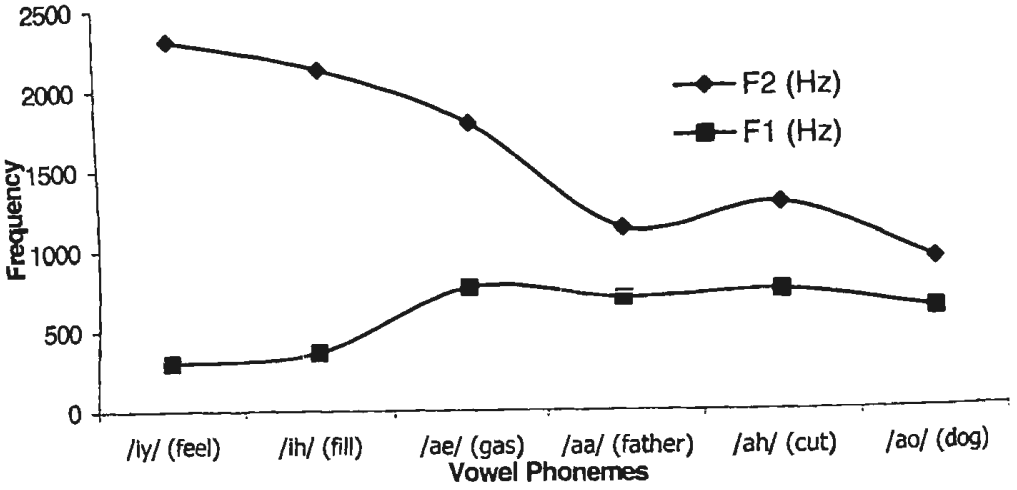


Figure 2.17 F1 and F2 values for articulations of some English vowels.

The major articulator for English vowels is the middle to rear portion of the tongue. The position of the tongue's surface is manipulated by large and powerful muscles in its root, which move it as a whole within the mouth. The linguistically important dimensions of movement are generally the ranges [front ⇔ back] and [high ⇔ low]. You can feel this movement easily. Say mentally, or whisper, the sound */iy/* (as in *see*) and then */aa/* (as in *father*). Do it repeatedly, and you will get a clear perception of the tongue movement from high to low. Now try */iy/* and then */uw/* (as in *blue*), repeating a few times. You will get a clear perception of place of articulation from front */iy/* to back */uw/*. Figure 2.18 shows a schematic characterization of English vowels in terms of relative tongue positions. There are two kinds of vowels: those in which tongue height is represented as a point and those in which it is represented as a vector.

Though the tongue hump is the major actor in vowel articulation, other articulators come into play as well. The most important secondary vowel mechanism for English and many other languages is lip rounding. Repeat the exercise above, moving from the */iy/* (*see*)

to the /uw/ (*blue*) position. Now rather than noticing the tongue movement, pay attention to your lip shape. When you say /iy/, your lips will be flat, slightly open, and somewhat spread. As you move to /uw/, they begin to *round out*, ending in a more puckered position. This lengthens the oral cavity during /uw/, and affects the spectrum in other ways.

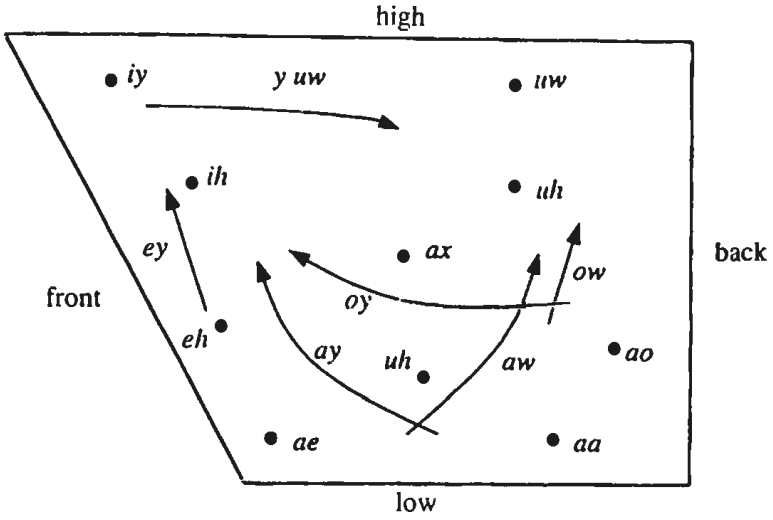


Figure 2.18 Relative tongue positions of English vowels [24].

Though there is always some controversy, linguistic study of phonetic abstractions, called *phonology*, has largely converged on the five binary features: +/- high, +/- low, +/- front, +/- back, and +/- round, plus the phonetically ambiguous but phonologically useful feature +/- tense, as adequate to uniquely characterize the major vowel distinctions of Standard English (and many other languages). Obviously, such a system is a little bit too free with logically contradictory specifications, such as [+high, +low], but these are excluded from real-world use. These features can be seen in Table 2.7.

Table 2.7 Phonological (abstract) feature decomposition of basic English vowels.

Vowel	high	low	front	back	round	tense
iy	+	-	+	-	-	+
ih	+	-	+	-	-	-
ae	-	+	+	-	-	+
aa	-	+	-	-	-	+
ah	-	-	-	-	-	+
ao	-	+	-	+	+	+
ax	-	-	-	-	-	-
eh	-	-	+	-	-	-
ow	-	-	-	+	+	+
uh	+	-	-	+	-	-
uw	+	-	-	+	-	+

This kind of abstract analysis allows researchers to make convenient statements about classes of vowels that behave similarly under certain conditions. For example, one may speak simply of the high vowels to indicate the set */iy, ih, uh, uw/*.

2.2.1.2. Consonants

Consonants, as opposed to vowels, are characterized by significant constriction or obstruction in the pharyngeal and/or oral cavities. Some consonants are voiced; others are not. Many consonants occur in pairs, that is, they share the same configuration of articulators, and one member of the pair additionally has voicing which the other lacks. One such pair is */s, z/*, and the voicing property that distinguishes them shows up in the non-periodic noise of the initial segment */s/* in Figure 2.5 as opposed to the voiced consonant end-phone, */z/*. Manner of articulation refers to the articulation mechanism of a consonant. The major distinctions in manner of articulation are listed in Table 2.8.

Table 2.8 Consonant manner of articulation.

Manner	Sample Phone	Example Words	Mechanism
Plosive	*/p/*	tat, tap	Closure in oral cavity
Nasal	*/m/*	team, meet	Closure of nasal cavity
Fricative	*/s/*	sick, kiss	Turbulent airstream noise
Retroflex liquid	*/r/*	rat, tar	Vowel-like, tongue high and curled back
Lateral liquid	*/l/*	lean, kneel	Vowel-like, tongue central, side airstream
Glide	*/y/,/w/*	yes, well	Vowel-like

The English phones that typically have voicing without complete obstruction or narrowing of the vocal tract are called *semivowels* and include */l, r/*, the *liquid* group, and */y, w/*, the *glide* group. Liquids, glides, and vowels are all *sonorant*, meaning they have continuous voicing. Liquids */l/* and */r/* are quite vowel-like and in fact may become *syllabic* or act entirely as vowels in certain positions, such as the *l* at the end of *edible*. In */l/*, the airstream flows around the sides of the tongue, leading to the descriptive term *lateral*. In */r/*, the tip of the tongue is curled back slightly, leading to the descriptive term *retroflex*. Figure 2.19 shows some semivowels.

Glides */y, w/* are basically vowels */iy, uw/* whose initial position within the syllable require them to be a little shorter and to lack the ability to be stressed, rendering them just different enough from true vowels that they are classed as a special category of consonant. Pre-vocalic glides that share the syllable-initial position with another consonant, such as the */y/* in the second syllable of *computer /k uh m . p y uw . t er/*, or the */w/* in *quick /k w ih k/*, are sometimes called *on-glides*. The semivowels, as a class, are sometimes called *approximants*, meaning that the tongue approaches the top of the oral cavity, but does not completely contact so as to obstruct the air flow.

Even the non-sonorant consonants that require complete or close-to-complete obstruction may still maintain some voicing before or during the obstruction, until the pressure dif-

ferential across the glottis starts to disappear, due to the closure. Such voiced consonants include */b,d,g, z, zh, v/*. They have a set of counterparts that differ only in their characteristic lack of voicing: */p,t,k, s, sh, f/*.

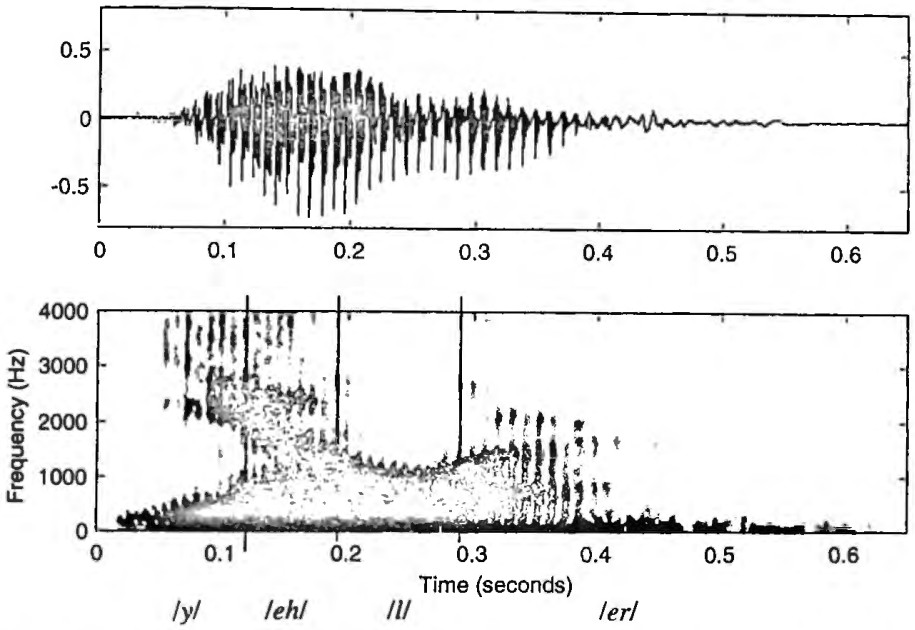


Figure 2.19 Spectrogram for the word *yeller*, showing semivowels */y/*, */l/*, */er/* (approximate phone boundaries shown with vertical lines).

Nasal consonants */m,n/* are a mixed bag: the oral cavity has significant constriction (by the tongue or lips), yet the voicing is continuous, like that of the sonorants, because, with the velar flap open, air passes freely through the nasal cavity, maintaining a pressure differential across the glottis.

A consonant that involves complete blockage of the oral cavity is called an obstruent stop, or plosive consonant. These may be voiced throughout if the trans-glottal pressure drop can be maintained long enough, perhaps through expansion of the wall of the oral cavity. In any case, there can be voicing for the early sections of stops. Voiced, unvoiced pairs of stops include: */b,p/*, */d,t/*, and */g,k/*. In viewing the waveform of a stop, a period of silence corresponding to the oral closure can generally be observed. When the closure is removed (by opening the constrictor, which may be lips or tongue), the trapped air rushes out in a more or less sudden manner. When the upper oral cavity is unimpeded, the closure of the vocal folds themselves can act as the initial blocking mechanism for a type of stop heard at the very beginning of vowel articulation in vowel-initial words like *atrophy*. This is called a *glottal stop*. Voiceless plosive consonants in particular exhibit a characteristic aperiodic *burst* of energy at the (articulatory) point of closure as shown in Figure 2.20 just prior to */i/*. By com-

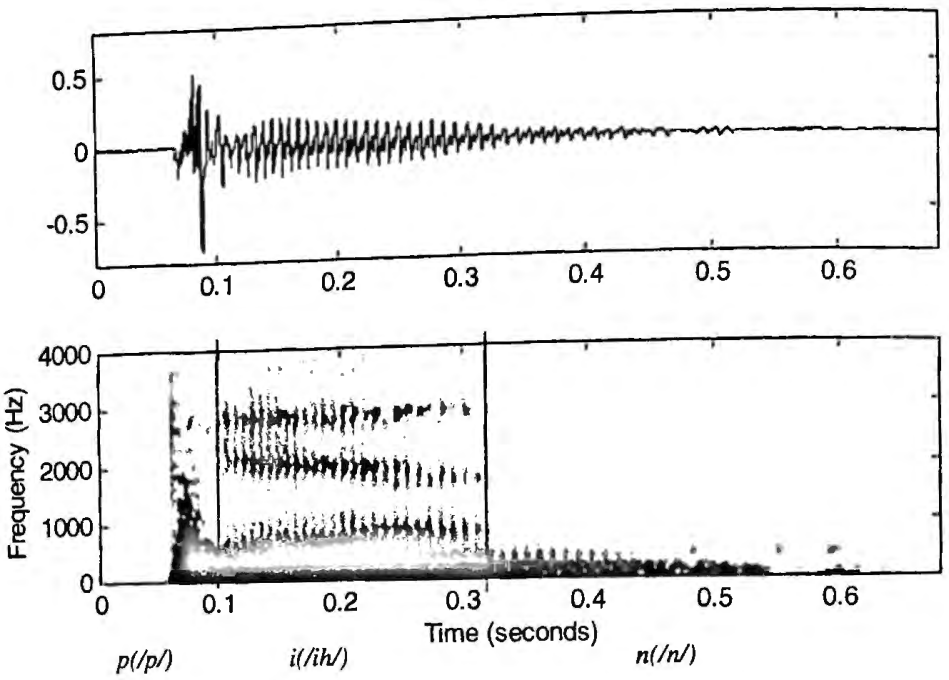


Figure 2.20 Spectrogram: stop release *burst* of /p/ in the word *pin*.

parison, the voicing of voiced plosive consonants may not always be obvious in a spectrogram.

A consonant that involves nearly complete blockage of some position in the oral cavity creates a narrow stream of turbulent air. The friction of this air stream creates a nonperiodic hiss-like effect. Sounds with this property are called fricatives and include /*s*, *z*/. There is no voicing during the production of *s*, while there can be voicing (in addition to the frication noise), during the production of *z*, as discussed above. /*s*, *z*/ have a common place of articulation, as explained below, and thus form a natural similarity class. Though controversial, /*h*/ can also be thought of as a (glottal) fricative. /*s*/ in word-initial position and /*z*/ in word-final position are exemplified in Figure 2.5.

Some sounds are complex combinations of manners of articulation. For example, the *affricates* consist of a stop (e.g., /*t*/), followed by a fricative [e.g., /*sh*/) combining to make a unified sound with rapid phases of closure and continuancy (e.g., {*t* + *sh*} = *ch* as in *church*). The affricates in English are the voiced/unvoiced pairs: /*j*/ (*d* + *zh*) and /*ch*/ (*t* + *sh*). The complete consonant inventory of English is shown in Table 2.9.

Consider the set /*m*/, /*n*/, /*ng*/ from Table 2.9. They are all voiced nasal consonants, yet they sound distinct to us. The difference lies in the location of the major constriction along the top of the oral cavity (from lips to velar area) that gives each consonant its unique quality. The articulator used to touch or approximate the given location is usually some spot along the length of the tongue. As shown in Figure 2.21, the combination of articulator and place of articulation gives each consonant its characteristic sound:

Table 2.9 Manner of articulation of English consonants.

Consonant Labels	Consonant Examples	Voiced?	Manner
b	**b**ig, able, tab	+	plosive
p	**p**ut, open, ta**p**	-	plosive
d	**d**ig, idea, wa**d**	+	plosive
t	**t**alk, sa**t**	-	plosive
g	**g**ut, angle, ta**g**	+	plosive
k	**c**ut, oa**k**en, ta**k**e	-	plosive
v	**v**at, over, ha**v**e	+	fricative
f	**f**ork, after, i**f**	-	fricative
z	**z**ap, lazy, ha**z**e	+	fricative
s	**s**it, ca**s**t, to**s**s	-	fricative
dh	**th**en, fa**th**er, scy**th**e	+	fricative
th	**th**in, no**th**ing, tru**th**	-	fricative
zh	genre, azure, bei**g**e	+	fricative
sh	**sh**e, cu**sh**ion, wa**sh**	-	fricative
jh	**j**oy, agile, e**dg**e	+	affricate
ch	**ch**in, ar**ch**er, ma**rch**	-	affricate
l	**l**id, el**b**ow, sa**il**	+	lateral
r	**r**ed, pa**r**t, fa**r**	+	retroflex
y	**y**acht, o**n**ion, **y**ard	+	glide
w	**w**ith, a**w**ay	+	glide
hh	**h**elp, a**h**ead, **h**otel	+	fricative
m	**m**at, a**m**id, a**im**	+	nasal
n	**n**o, e**n**d, pa**n**	+	nasal
ng	sing, ang**er**, dri**nk**	+	nasal

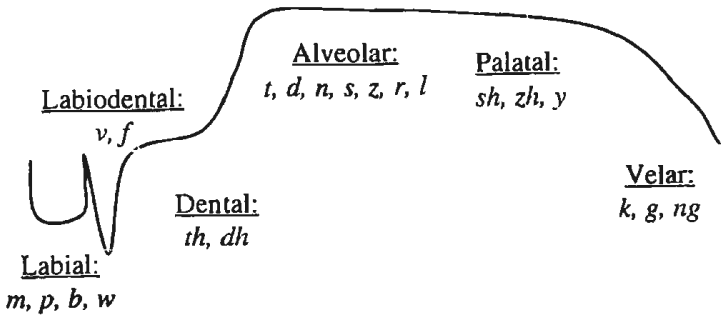


Figure 2.21 The major places of consonant articulation with respect to the human mouth.

- The *labial* consonants have their major constriction at the lips. This includes */p/, /b/* (these two differ only by manner of articulation) and */m/* and */w/*.
- The class of *dental or labio-dental* consonants includes */f, v/* and */th, dh/* (the members of these groups differ in manner, not place).
- *Alveolar* consonants bring the front part of the tongue, called the tip or the part behind the tip called the blade, into contact or approximation to the alveolar ridge, rising semi-vertically above and behind the teeth. These include */t, d, n, s, z, r, l/*. The members of this set again differ in manner of articulation (voicing, continuity, nasality), rather than place.
- *Palatal* consonants have approximation or constriction on or near the roof of the mouth, called the palate. The members include */sh, zh, y/*.
- *Velar* consonants bring the articulator (generally the back of the tongue), up to the rearmost top area of the oral cavity, near the velar flap. Velar consonants in English include */k, g/* (differing by voicing) and the nasal continuant */ng/*.

With the place terminology, we can complete the descriptive inventory of English consonants, arranged by manner (rows), place (columns), and voiceless/voiced (pairs in cells) as illustrated in Table 2.10.

Table 2.10 The consonants of English arranged by place (columns) and manner (rows).

	Labial	**Labio-dental**	**Dental**	**Alveolar**	**Palatal**	**Velar**	**Glottal**
Plosive	*p b*			*t d*		*k g*	*?*
Nasal	*m*			*n*		*ng*	
Fricative		*f v*	*th dh*	*s z*	*sh zh*		*h*
Retroflex sonorant				*r*			
Lateral sonorant				*l*			
Glide	*w*				*y*		

2.2.1.3. Phonetic Typology

The oral, nasal, pharyngeal, and glottal mechanisms actually make available a much wider range of effects than English happens to use. So, it is expected that other languages would utilize other vocal mechanisms, in an internally consistent but essentially arbitrary fashion, to represent their lexicons. In addition, often a vocal effect that is part of the systematic linguistic phonetics of one language is present in others in a less codified, but still perceptible, form. For example, Japanese vowels have a characteristic distinction of length that can be hard for non-natives to perceive and to use when learning the language. The words *kado* (*corner*) and *kaado* (*card*) are spectrally identical, differing only in that *kado* is much shorter

in all contexts. The existence of such minimally-contrasting pairs is taken as conclusive evidence that *length* is phonemically distinctive for Japanese. As noted above, what is linguistically distinctive in any one language is generally present as a less *meaningful* signaling dimension in other languages. Thus, vowel length can be manipulated in any English word as well, but this occurs either consciously for emphasis or humorous effect, or unconsciously and very predictably at clause and sentence end positions, rather than to signal lexical identity in all contexts, as in Japanese.

Other interesting sounds that the English language makes no linguistic use of include the trilled *r* sound and the implosive. The trilled *r* sound is found in Spanish, distinguishing (for example) the words *pero* (*but*) and *perro* (*dog*). This trill could be found in times past as a non-lexical sound used for emphasis and interest by American circus ringmasters and other showpersons.

While the world's languages have all the variety of manner of articulation exemplified above and a great deal more, the primary dimension lacking in English that is exploited by a large subset of the world's languages is pitch variation. Many of the huge language families of Asia and Africa are tonal, including all varieties of Chinese. A large number of other languages are not considered strictly tonal by linguistics, yet they make systematic use of pitch contrasts. These include Japanese and Swedish. To be considered tonal, a language should have lexical meaning contrasts cued by pitch, just as the lexical meaning contrast between *pig* and *big* is cued by a voicing distinction in English. For example, Mandarin Chinese has four primary tones (tones can have minor context-dependent variants just like ordinary phones, as well) as shown in Table 2.11.

Table 2.11 The contrastive tones of Mandarin Chinese.

Tone	Shape	Example	Chinese	Meaning
1	High level	*ma*	妈	mother
2	High rising	*ma*	麻	numb
3	Low rising	*ma*	马	horse
4	High falling	*ma*	骂	to scold

Though English does not make systematic use of pitch in its inventory of word contrasts, nevertheless, as we always see with any possible phonetic effect, pitch is systematically varied in English to signal a speaker's emotions, intentions, and attitudes, and it has some linguistic function in signaling grammatical structure as well. Pitch variation in English will be considered in more detail in Chapter 15.

2.2.2. The Allophone: Sound and Context

The vowel and consonant charts provide abstract symbols for the phonemes – major sound distinctions. Phonemic units should be correlated with potential meaning distinctions. For example, the change created by holding the tongue high and front (*/iy/*) vs. directly down

from the (frontal) position for */eh/*, in the consonant context */m _ n/*, corresponds to an important meaning distinction in the lexicon of English: *mean /m iy n/* vs. *men /m eh n/*. This meaning contrast, conditioned by a pair of rather similar sounds, in an identical context, justifies the inclusion of */iy/* and */eh/* as logically separate distinctions.

However, one of the fundamental, meaning-distinguishing sounds is often modified in some systematic way by its phonetic neighbors. The process by which neighboring sounds influence one another is called *coarticulation*. Sometimes, when the variations resulting from coarticulatory processes can be consciously perceived, the modified phonemes are called *allophones*. Allophonic differences are always *categorical*, that is, they can be understood and denoted by means of a small, bounded number of symbols or diacritics on the basic phoneme symbols.

As an experiment, say the word *like* to yourself. Feel the front of the tongue touching the alveolar ridge (cf. Figure 2.21) when realizing the initial phoneme */l/*. This is one allophone of */l/*, the so-called *light* or *clear /l/*. Now say *kill*. In this word, most English speakers will no longer feel the front part of the tongue touch the alveolar ridge. Rather, the */l/* is realized by stiffening the broad midsection of the tongue in the rear part of the mouth while the continuant airstream escapes laterally. This is another allophone of */l/*, conditioned by its syllable-final position, called the *dark /l/*. Predictable contextual effects on the realization of phones can be viewed as a nuisance for speech recognition, as will be discussed in Chapter 9. On the other hand, such variation, because it is systematic, could also serve as a cue to the syllable, word, and prosodic structure of speech.

Now experiment with the sound */p/* by holding a piece of tissue in front of your mouth while saying the word *pin* in a normal voice. Now repeat this experiment with *spin*. For most English speakers, the word *pin* produces a noticeable puff of air, called aspiration. But the same phoneme, */p/*, embedded in the consonant cluster */sp/* loses its aspiration (burst, see the lines bracketing the */p/* release in *pin* and *spin* in Figure 2.22), and because these two types of */p/* are in complementary distribution (completely determined by phonetic and syllabic context), the difference is considered allophonic.

Try to speak the word *bat* in a framing phrase *say bat again*. Now speak *say bad again*. Can you feel the length difference in the vowel */ae/*? A vowel before a voiced consonant, e.g., */d/*, seems typically longer than the same vowel before the unvoiced counterpart, in this case */t/*.

A sound phonemicized as */t/* or */d/*, that is, a stop made with the front part of the tongue, may be reduced to a quick tongue tap that has a different sound than either */t/* or */d/* in fuller contexts. This process is called flapping. It occurs when */t/* or */d/* closes a stressed vowel (coda position) followed by an unstressed vowel, as in: *bitter, batter, murder, quarter, humidity,* and can even occur across words as long as the preconditions are met, as in *you can say that again*. Sometimes the velar flap opens too soon (anticipation), giving a characteristically nasal quality to some pre-nasal vowels such as */ae/* in *ham* vs. *had*. We have a more detailed discussion on allophones in Chapter 9.

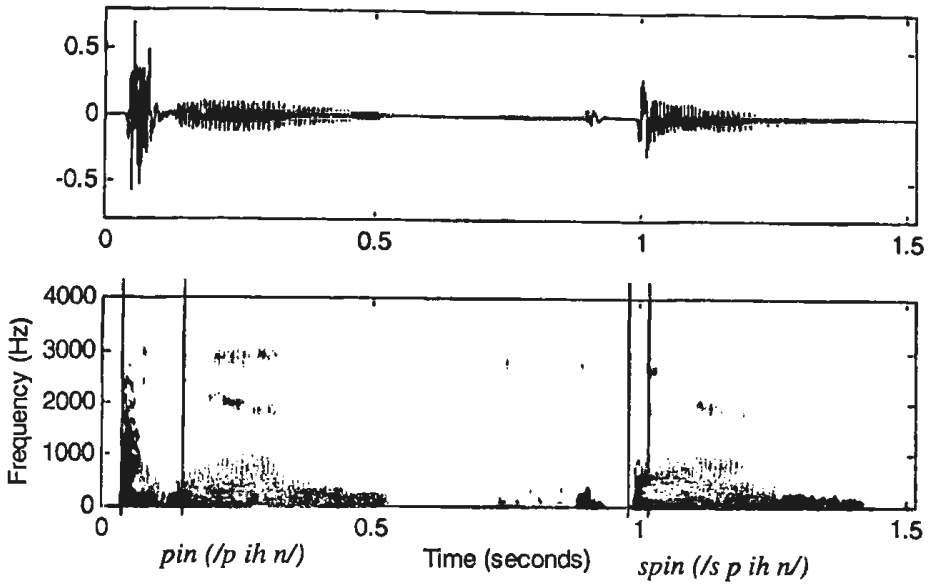


Figure 2.22 Spectrogram: bursts of *pin* and *spin*. The relative duration of a *p*-burst in different phonetic contexts is shown by the differing width of the area between the vertical lines.

2.2.3. Speech Rate and Coarticulation

In addition to allophones, there are other variations in speech for which no small set of established categories of variation can be established. These are *gradient*, existing along a scale for each relevant dimension, with speakers scattered widely. In general, it is harder to become consciously aware of coarticulation effects than of allophonic alternatives.

Individual speakers may vary their rates according to the content and setting of their speech, and there may be great inter-speaker differences as well. Some speakers may pause between every word, while others may speak hundreds of words per minute with barely a pause between sentences. At the faster rates, formant targets are less likely to be fully achieved. In addition, individual allophones may merge.

For example [20], consider the utterance *Did you hit it to Tom*? The pronunciation of this utterance is */d ih d y uw h ih t ih t t uw t aa m/*. However, a realistic, casual rendition of this sentence would appear as */d ih jh ax hh ih dx ih t ix t aa m/*, where */ix/* is a reduced schwa */ax/* that is short and often unvoiced, and */dx/* is a kind of shortened, indistinct stop, intermediate between */d/* and */t/*. The following five phonologic rules have operated on altering the pronunciation in the example:

- Palatalization of */d/* before */y/* in *did you*
- Reduction of unstressed */u/* to schwa in *you*

- Flapping of intervocalic /t/ in *hit it*
- Reduction of schwa and devoicing of /u/ in *to*
- Reduction of geminate (double consonant) /t/ in *it to*

There are also coarticulatory influences in the spectral appearance of speech sounds, which can only be understood at the level of spectral analysis. For example, in vowels, consonant neighbors can have a big effect on formant trajectories near the boundary. Consider the differences in *F1* and *F2* in the vowel */eh/* as realized in words with different initial consonants *bet*, *debt*, and *get*, corresponding to the three major places of articulation (labial, alveolar, and velar), illustrated in Figure 2.23. You can see the different relative spreads of *F1* and *F2* following the initial stop consonants.

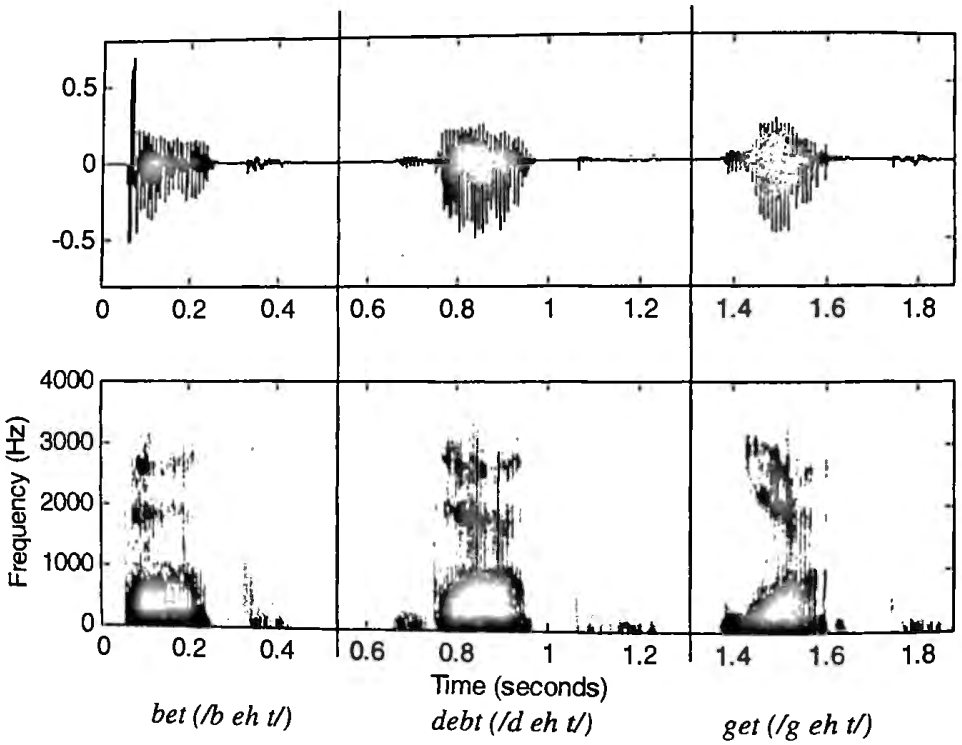


Figure 2.23 Spectrogram: *bet*, *debt*, and *get* (separated by vertical lines). Note different relative spreads of *F1* and *F2* following the initial stop consonants in each word.

Now let's see different consonants following the same vowel, *ebb*, *head*, and *egg*. In Figure 2.23, the coarticulatory effect is *perseverance*; i.e., in the early part of the vowel the articulators are still somewhat set from realization of the initial consonant. In the *ebb*, *head*, and *egg* examples shown in Figure 2.24, the coarticulatory effect is *anticipation*; i.e., in the latter part of the vowel the articulators are moving to prepare for the upcoming consonant articulation. You can see the increasing relative spread of *F1* and *F2* at the final vowel-consonant transition in each word.

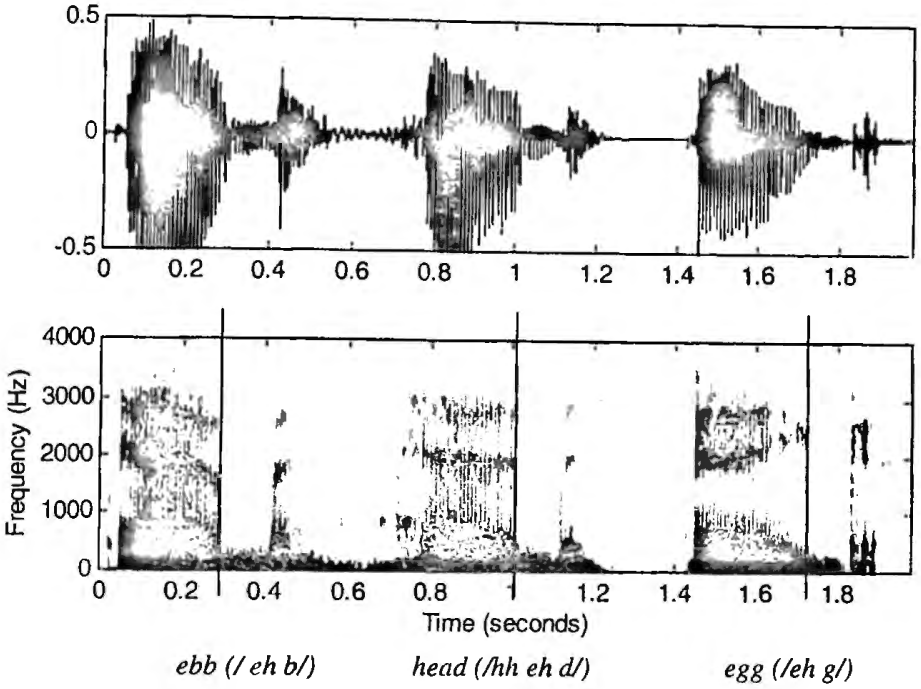


Figure 2.24 Spectrogram: *ebb*, *head*, and *egg*. Note the increasing relative spread of *F1* and *F2* at the final vowel-consonant transition in each word.

2.3. SYLLABLES AND WORDS

Phonemes are small building blocks. To contribute to language meaning, they must be organized into longer cohesive spans, and the units so formed must be combined in characteristic patterns to be meaningful, such as syllables and words in the English language.

2.3.1. Syllables

An intermediate unit, the *syllable*, is sometimes thought to interpose between the phones and the word level. The syllable is a slippery concept, with implications for both production and perception. Here we will treat it as a perceptual unit. Syllables are generally centered around vowels in English, giving two perceived syllables in a word like *tomcat*: */tOm-cAt/*. To completely parse a word into syllables requires making judgments of consonant affiliation (with the syllable peak vowels). The question of whether such judgments should be based on articulatory or perceptual criteria, and how they can be rigorously applied, remains unresolved.

Syllable centers can be thought of as *peaks* in sonority (high-amplitude, periodic sections of the speech waveform). These sonority peaks have affiliated *shoulders* of strictly non-increasing sonority. A scale of sonority can be used, ranking consonants along a continuum of stops, affricates, fricatives, and approximants. So, in a word like *verbal*, the syllabification would be *ver-bal*, or *verb-al*, but not *ve-rbal*, because putting the approximant */r/* before the stop */b/* in the second syllable would violate the non-decreasing sonority requirement heading into the syllable.

As long as the sonority conditions are met, the exact affiliation of a given consonant that could theoretically affiliate on either side can be ambiguous, unless determined by higher-order considerations of word structure, which may block affiliation. For example, in a word like *beekeeper*, an abstract boundary in the compound between the component words *bee* and *keeper* keeps us from accepting the syllable parse: *beek-eeper*, based on lexical interpretation. However, the same phonetic sequence in *beaker* could, depending on one's theory of syllabicity, permit affiliation of the *k: beak-er*. In general, the syllable is a unit that has intuitive plausibility but remains difficult to pin down precisely.

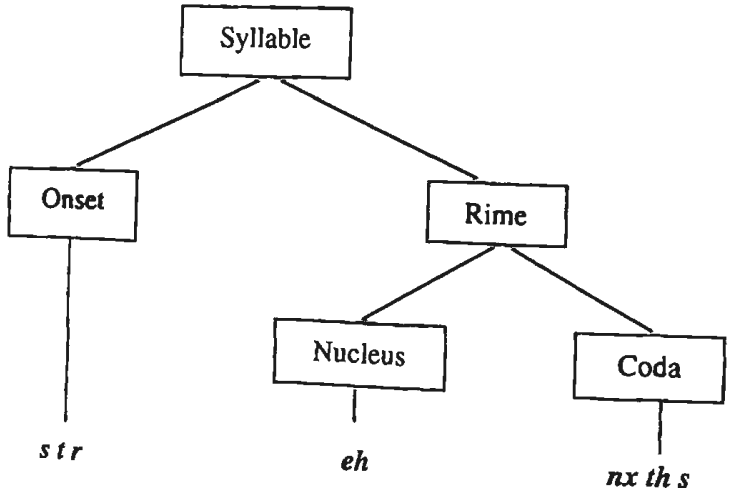


Figure 2.25 The word/syllable *strengths* (*/s t r eh nx th s/*) is the longest syllable of English.

Syllables are thought (by linguistic theorists) to have internal structure, and the terms used are worth knowing. Consider a big syllable such as *strengths /s t r eh nx th s/*. This consists of a vowel peak, called the *nucleus*, surrounded by the other sounds in characteristic positions. The *onset* consists of initial consonants if any, and the rime is the nucleus with trailing consonants (the part of the syllable that matters in determining poetic rhyme). The coda consists of consonants in the rime following the nucleus (in some treatments, the last consonant in a final cluster would belong to an *appendix*). This can be diagrammed as a syllable parse tree as shown in Figure 2.25. The syllable is sometimes thought to be the primary domain of coarticulation, that is, sounds within a syllable influence one another's realization more than the same sounds separated by a syllable boundary.

2.3.2. Words

The concept of words seems intuitively obvious to most speakers of Indo-European languages. It can be loosely defined as a lexical item, with an agreed-upon meaning in a given speech community, that has the freedom of syntactic combination allowed by its type (noun, verb, etc.).

In spoken language, there is a segmentation problem: words *run together* unless affected by a disfluency (unintended speech production problem) or by the deliberate placement of a pause (silence) for some structural or communicative reason. This is surprising to many people, because literacy has conditioned speakers/readers of Indo-European languages to expect a *blank space* between words on the printed page. But in speech, only a few true pauses (the aural equivalent of a blank space) may be present. So, what appears to the reading eye as *never give all the heart, for love* would appear to the ear, if we simply use letters to stand for their corresponding English speech sounds, as *nevergivealltheheart forlove* or, in phonemes, as *n eh v er g ih v ah l dh ax h aa r t \\ f ao r l ah v*. The \\ symbol marks a linguistically motivated pause, and the units so formed are sometimes called *intonation phrases*, as explained in Chapter 15.

Certain facts about word structure and combinatorial possibilities are evident to most native speakers and have been confirmed by decades of linguistic research. Some of these facts describe relations among words when considered in isolation, or concern groups of related words that seem intuitively similar along some dimension of form or meaning – these properties are *paradigmatic*. Paradigmatic properties of words include part-of-speech, inflectional and derivational morphology, and compound structure. Other properties of words concern their behavior and distribution when combined for communicative purposes in fully functioning utterances – these properties are *syntagmatic*.

2.3.2.1. Lexical Part-of-Speech

Lexical part-of-speech (POS) is a primitive form of linguistic theory that posits a restricted inventory of word-type categories, which capture generalizations of word forms and distributions. Assignment of a given POS specification to a word is a way of summarizing certain facts about its potential for syntagmatic combination. Additionally, paradigms of word formation processes are often similar within POS types and subtypes as well. The word properties upon which POS category assignments are based may include affixation behavior, very abstract semantic typologies, distributional patterns, compounding behavior, historical development, productivity and generalizabilty, and others.

A typical set of POS categories would include *noun*, *verb*, *adjective*, *adverb*, *interjection*, *conjunction*, *determiner*, *preposition*, and *pronoun*. Of these, we can observe that certain classes of words consist of infinitely large membership. This means new members can be added at any time. For example, the category of noun is constantly expanded to accommodate new inventions, such as Velcro or Spandex. New individuals are constantly being born, and their names are a type of noun called *proper noun*. The proliferation of words us-

ing the descriptive prefix *cyber* is another recent set of examples: *cyberscofflaw, cybersex,* and even *cyberia* illustrate the infinite creativity of humans in manipulating word structure to express new shades of meaning, frequently by analogy with, and using fragments of, existing vocabulary. Another example is the neologism *sheeple*, a noun combining the forms and meanings of *sheep* and *people* to refer to large masses of people who lack the capacity or willingness to take independent action. We can create new words whenever we like, but they had best fall within the predictable paradigmatic and syntagmatic patterns of use summarized by the existing POS generalizations, or there will be little hope of their adoption by any other speaker. These open POS categories are listed in Table 2.12. Nouns are inherently referential. They refer to persons, places, and things. Verbs are predicative; they indicate relations between entities and properties of entities, including participation in events. Adjectives typically describe and more completely specify noun reference, while adverbs describe, intensify, and more completely specify verbal relations. Open-class words are sometimes called *content* words, for their referential properties.

Table 2.12 Open POS categories.

Tag	Description	Function	Example
N	Noun	Names entity	*cat*
V	Verb	Names event or condition	*forget*
Adj	Adjective	Descriptive	*yellow*
Adv	Adverb	Manner of action	*quickly*
Interj	Interjection	Reaction	*oh!*

In contrast to the open-class categories, certain other categories of words only rarely and very slowly admit new members over the history of English development. These closed POS categories are shown in Table 2.13. The closed-category words are fairly stable over time. Conjunctions are used to join larger syntactically complete phrases. Determiners help to narrow noun reference possibilities. Prepositions denote common spatial and temporal relations of objects and actions to one another. Pronouns provide a convenient substitute for noun phrases that are fully understood from context. These words denote grammatical relations of other words to one another and fundamental properties of the world and how humans understand it. They can, of course, change slowly; for example, the Middle English pronoun *thee* is no longer in common use. The closed-class words are sometimes called *function* words.

Table 2.13 Closed POS categories.

Tag	Description	Function	Example
Conj	Conjunction	Coordinates phrases	*and*
Det	Determiner	Indicates definiteness	*the*
Prep	Preposition	Relations of time, space, direction	*from*
Pron	Pronoun	Simplified reference	*she*

The set of POS categories can be extended indefinitely. Examples can be drawn from the Penn Treebank project (http://www.cis.upenn.edu/ldc) as shown in Table 2.14, where you can find the proliferation of sub-categories, such as *Verb, base form* and *Verb, past tense*. These categories incorporate *morphological* attributes of words into the POS label system discussed in Section 2.3.2.2.

Table 2.14 Treebank POS categories – an expanded inventory.

String	Description	Example
CC	Coordinating conjunction	and
CD	Cardinal number	two
DT	Determiner	the
EX	Existential *there*	there (*There was an old lady*)
FW	Foreign word	omerta
IN	Preposition, subord. conjunction	over, but
JJ	Adjective	yellow
JJR	Adjective, comparative	better
JJS	Adjective, superlative	best
LS	List item marker	
MD	Modal	might
NN	Noun, singular or mass	rock, water
NNS	Noun, plural	rocks
NNP	Proper noun, singular	Joe
NNPS	Proper noun, plural	Red Guards
PDT	Predeterminer	all (*all the girls*)
POS	Possessive ending	's
PRP	Personal pronoun	I
PRP$	Possessive pronoun	mine
RB	Adverb	quickly
RBR	Adverb, comparative	higher (*shares closed higher.*)
RBS	Adverb, superlative	highest (*he jumped highest of all.*)
RP	Particle	up (*take up the cause*)
TO	*to*	to
UH	Interjection	hey!
VB	Verb, base form	choose
VBD	Verb, past tense	chose
VBG	Verb, gerund, or present participle	choosing
VBN	Verb, past participle	chosen
VBP	Verb, non-third person sing. present	jump
VBZ	Verb, third person singular present	jumps
WDT	Wh-determiner	which
WP	Wh-pronoun	who
WP$	Possessive wh-pronoun	whose
WRB	Wh-adverb	when (*When he came, it was late.*)

POS tagging is the process of assigning a part-of-speech or other lexical class marker to each word in a corpus. There are many algorithms to automatically tag input sentences into a set of tags. Rule-based methods [45], hidden Markov models (see Chapter 8) [23, 29, 46], and machine-learning methods [6] are used for this purpose.

2.3.2.2. Morphology

Morphology is about the subparts of words, i.e., the patterns of word formation including inflection, derivation, and the formation of compounds. English mainly uses prefixes and suffixes to express *inflection* and *derivational* morphology.

Inflectional morphology deals with variations in word form that reflect the contextual situation of a word in phrase or sentence syntax, and that rarely have direct effect on interpretation of the fundamental meaning expressed by the word. English inflectional morphology is relatively simple and includes person and number agreement and tense markings only. The variation in *cats* (vs. *cat*) is an example. The plural form is used to refer to an indefinite number of cats greater than one, depending on a particular situation. But the basic POS category (*noun*) and the basic meaning (*felis domesticus*) are not substantially affected. Words related to a common lemma via inflectional morphology are said to belong to a common paradigm, with a single POS category assignment. In English, common paradigm types include the verbal set of affixes (pieces of words): *-s, -ed, -ing*; the noun set: *-s*; and the adjectival *-er, -est*. Note that sometimes the base form may change spelling under affixation, complicating the job of automatic textual analysis methods. For historical reasons, certain paradigms may consist of highly idiosyncratic irregular variation as well, e.g., *go, going, went, gone* or *child, children*. Furthermore, some words may belong to defective paradigms, where only the singular (noun: *equipment*) or the plural (noun: *scissors*) is provided for.

In *derivational morphology*, a given root word may serve as the source for wholly new words, often with POS changes as illustrated in Table 2.15. For example, the terms *racial* and *racist*, though presumably based on a single root word *race*, have different POS possibilities (*adjective* vs. *noun-adjective*) and meanings. Derivational processes may induce pronunciation change or stress shift (e.g., *electric* vs. *electricity*). In English, typical derivational affixes (pieces of words) that are highly productive include prefixes and suffixes: *re-, pre-, -ial, -ism, -ish, -ity, -tion, -ness, -ment, -ious, -ify, -ize*, and others. In many cases, these can be added successively to create a complex layered form.

Table 2.15 Examples of stems and their related forms across POS categories.

Noun	Verb	Adjective	Adverb
criticism	*criticize*	*critical*	*critically*
fool	*fool*	*foolish*	*foolishly*
industry, industrialization	*industrialize*	*industrial,industrious*	*industriously*
employ, employee, employer	*employ*	*employable*	*employably*
certification	*certify*	*certifiable*	*certifiably*

Generally, word formation operates in layers, according to a kind of word syntax: *(deriv-prefix)* root (root)* (deriv-suffix)* (infl-suffix).* This means that one or more *roots* can be compounded in the inner layer, with one or more optional *derivational prefixes*, followed by any number of optional *derivational suffixes*, capped off with no more than one *inflectional suffix*. There are, of course, limits on word formation, deriving both from semantics of the component words and simple lack of imagination. An example of a nearly maximal word in English might be *autocyberconceptualizations*, meaning (perhaps!) multiple instances of automatically creating computer-related concepts. This word lacks only compounding to be truly maximal. This word has a derivational prefix *auto-*, two root forms compounded (*cyber* and *concept*, though some may prefer to analyze *cyber-* as a prefix), three derivational suffixes (*-ual, -ize, -ation*), and is capped off with the plural inflectional suffix for nouns, *-s*.

2.3.2.3. Word Classes

POS classes are based on traditional grammatical and lexical analysis. With improved computational resources, it has become possible to examine words in context and assign words to groups according to their actual behavior in real text and speech from a statistical point of view. These kinds of classifications can be used in language modeling experiments for speech recognition, text analysis for text-to-speech synthesis, and other purposes.

One of the main advantages of word classification is its potential to derive more refined classes than traditional POS, while only rarely actually crossing traditional POS group boundaries. Such a system may group words automatically according to the similarity of usage with respect to their word neighbors. Consider classes automatically found by the classification algorithms of Brown *et al.* [7]:

{Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends}
{great big vast sudden mere sheer gigantic lifelong scant colossal}
{down backwards ashore sideways southward northward overboard aloft adrift}
{mother wife father son husband brother daughter sister boss uncle}
{John George James Bob Robert Paul William Jim David Mike}
{feet miles pounds degrees inches barrels tons acres meters bytes}

You can see that words are grouped together based on the semantic meaning, which is different from word classes created purely from syntactic point of view. Other types of classification are also possible, some of which can identify semantic relatedness across traditional POS categories. Some of the groups derived from this approach may include follows:

{problems problem solution solve analyzed solved solving}
{write writes writing written wrote pen}
{question questions asking answer answers answering}
{published publication author publish writer titled}

2.4. SYNTAX AND SEMANTICS

Syntax is the study of the patterns of formation of sentences and phrases from words and the rules for the formation of grammatical sentences. Semantics is another branch of linguistics dealing with the study of meaning, including the ways meaning is structured in language and changes in meaning and form over time.

2.4.1. Syntactic Constituents

Constituents represent the way a sentence can be divided into its grammatical subparts as constrained by common grammatical patterns (which implicitly incorporate normative judgments on acceptability). Syntactic constituents at least respect, and at best explain, the linear order of words in utterances and text. In this discussion, we will not strictly follow any of the many theories of syntax but will instead bring out a few basic ideas common to many approaches. We will not attempt anything like a complete presentation of the grammar of English but instead focus on a few simple phenomena.

Most work in syntactic theory has adopted machinery from traditional grammatical work on written language. Rather than analyze toy sentences, let's consider what kinds of superficial syntactic patterns are lurking in a random chunk of serious English text, excerpted from David Thoreau's essay *Civil Disobedience* [43]:

The authority of government, even such as I am willing to submit to – for I will cheerfully obey those who know and can do better than I, and in many things even those who neither know nor can do so well – is still an impure one: to be strictly just, it must have the sanction and consent of the governed. It can have no pure right over my person and property but what I concede to it. The progress from an absolute to a limited monarchy, from a limited monarchy to a democracy, is a progress toward a true respect for the individual.

2.4.1.1. Phrase Schemata

Words may be combined to form phrases that have internal structure and unity. We use generalized schemata to describe the phrase structure. The goal is to create a simple, uniform template that is independent of POS category.

Let's first consider nouns, a fundamental category referring to persons, places, and things in the world. The noun and its immediate modifiers form a constituent called the noun phrase (*NP*). To generalize this, we consider a word of arbitrary category, say category *X* (which could be a noun *N* or a verb *V*). The generalized rule for a phrase *XP* is $XP \Rightarrow$ *(modifiers) X-head (post-modifiers)*, where *X* is the head, since it dominates the configuration and names the phrase. Elements preceding the head in its phrase are *premodifiers* and elements following the head are *postmodifiers*. *XP*, the culminating phrase node, is called a *maximal projection* of category *X*. We call the whole structure an *x-template*. Maximal projections, *XP*, are the primary currency of basic syntactic processes. The post-modifiers are usually maximal projections (another head, with its own post-modifiers forming an *XP* on its own) and are sometimes termed *complements*, because they are often required by the lexical properties of the head for a complete meaning to be expressed (e.g., when *X* is a preposition

or verb). Complements are typically noun phrases (*NP*), prepositional phrases (*PP*), verb phrases (*VP*), or sentence/clause (*S*), which make an essential contribution to the head's reference or meaning, and which the head requires for semantic completeness. Premodifiers are likely to be adverbs, adjectives, quantifiers, and determiners, i.e., words that help to specify the meaning of the head but may not be essential for completing the meaning. With minor variations, the *XP* template serves for most phrasal types, based on the POS of the head (*N, V, ADJ*, etc.).

For *NP*, we thus have *NP* $\Rightarrow$ *(det) (modifier) **head-noun** (post-modifier)*. This rule describes an *NP* (noun phrase – left side of arrow) in terms of its optional and required internal contents (right side of the arrow). *Det* is a word like *the* or *a* that helps to resolve the reference to a specific or an unknown instance of the noun. The *modifier* gives further information about the noun. The *head* of the phrase, and the only mandatory element, is the noun itself. *Post-modifiers* also give further information, usually in a more elaborate syntactic form than the simpler pre-modifiers, such as a relative clause or a prepositional phrase (covered below). The noun phrases of the passage above can be parsed as shown in Table 2.16. The head nouns may be personal pronouns (*I, it*), demonstrative and relative pronouns (*those*), coordinated nouns (*sanction and consent*), or common nouns (*individual*). The modifiers are mostly adjectives (*impure, pure*) or verbal forms functioning as adjectives (*limited*). The post-modifiers are interesting, in that, unlike the (pre-)modifiers, they are typically full phrases themselves, rather than isolated words. They include relative clauses (which are a kind of dependent sentence, e.g., *[those] who know and can do better than I*), as well as prepositional phrases (*of the governed*).

Table 2.16 *NP*s of the sample passage.

NP	Det	Mod	Head Noun	Post-Mod
1	the		**authority**	of government
2		even	**such**	as I am willing to submit to
3			**I**	
4			**those**	who know and can do better than I
5		many	**things**	
6		even	**those**	who neither know nor can do so well
7	an	impure	**one**	
8			**it**	
9	the		**sanction and consent**	of the governed
10	no	pure	**right**	over my person ... concede to it.
11	the		**progress**	from an absolute to a limited monarchy
12	an	absolute	**[monarchy]**	
13	a	limited	**monarchy**	
14	a		**democracy**	
15	a		**progress**	
16	a	true	**respect**	for the individual
17	the		**individual**	

Table 2.17 *PP*s of the sample passage.

Head Prep	Complement (Postmodifier)
of	Government
as	I am willing to submit to
than	I
in	many things
of	the governed
over	my person and property
to	it
from	an absolute [monarchy]
to	a limited monarchy
to	a democracy
toward	a true respect [for the individual]
for	the individual

Prepositions express spatial and temporal relations, among others. These are also said to project according to the *X*-template, but usually lack a pre-modifier. Some examples from the sample passage are listed in Table 2.17. The complements of *PP* are generally *NP*s, which may be simple head nouns like *government*. However, other complement types, such as the verb phrase in *after discussing it with Jo*, are also possible.

For verb phrases, the postmodifier (or complement) of a head verb would typically be one or more *NP* (noun phrase) maximal projections, which might, for example, function as a direct object in a *VP* like *pet the cat*. The complement may or may not be optional, depending on characteristics of the head. We can now make some language-specific generalizations about English. Some verbs, such as *give*, may take more than one kind of complement. So an appropriate template for a *VP* maximal projection in English would appear abstractly as *VP* $\Rightarrow$ *(modifier) verb (modifier) (Complement1, Complement2 ComplementN).* Complements are usually regarded as maximal projections, such as *NP*, *ADJP*, etc., and are enumerated in the template above, to cover possible multi-object verbs, such as *give*, which take both direct and indirect objects. Certain types of adverbs (*really*, *quickly*, *smoothly*, etc.) could be considered fillers for the *VP* modifier slots (before and after the head). In the sample passage, we find the following verb phrases as shown in Table 2.18.

VP presents some interesting issues. First, notice the multi-word verb *submit to*. Multi-word verbs such as *look after* and *put up with* are common. We also observe a number of auxiliary elements clustering before the verb in sentences of the sample passage: *am willing to submit to*, *will cheerfully obey*, and *can do better*. Rather than considering these as simple modifiers of the verbal head, they can be taken to have *scope* over the *VP* as a whole, which implies they are outside the *VP*. Since they are outside the *VP*, we can assume them to be heads in their own right, of phrases which require a *VP* as their complement. These elements mainly express tense (time or duration of verbal action) and modality (likelihood or probability of verbal action). In a full sentence, the *VP* has explicit or implicit inflection (projected from its verbal head) and indicates the person, number, and other context-dependent

features of the verb in relation to its arguments. In English, the person (first, second, third) and number (singular, plural) attributes, collectively called agreement features, of subject and verb must match. For simplicity, we will lump all these considerations together as inflectional elements, and posit yet another phrase type, the Inflectional Phrase (*IP*): *IP* ⇒ *premodifier head VP-complement.*

Table 2.18 *VP*s of the sample passage.

Pre-mod	Verb Head	Post-mod	Complement
	submit to		[the authority of government]
cheerfully	**obey**		those who know and can do better than I
	is	still	an impure one
	be		strictly just
	have		the sanction
	have		no pure right
	concede		to it
	is		a progress

The premodifier slot (sometimes called the *specifier* position in linguistic theory) of an *IP* is often filled by the subject of the sentence (typically a noun or *NP*). Since the *IP* unites the subject of a sentence with a *VP*, *IP* can also be considered simply as the sentence category, often written as *S* in speech grammars.

2.4.1.2. Clauses and Sentences

The *subject* of a sentence is what the sentence is mainly about. A *clause* is any phrase with both a subject and a *VP* (*predicate* in traditional grammars) that has potentially independent interpretation – thus, for us, a clause is an *IP*, a kind of sentence. A phrase is a constituent lacking either subject, predicate, or both. We have reviewed a number of phrase types above. There are also various types of clauses and sentences.

Even though clauses are sentences from an internal point of view (having subject and predicate), they often function as simpler phrases or words would, e.g., as modifiers (adjective and adverbs) or nouns and noun phrases. Clauses may appear as post-modifiers for nouns (so-called *relative clauses*), basically a kind of adjective clause, sharing their subjects with the containing sentence. Some clauses function as *NP*s in their own right. One common clause type substitutes a *wh-word* like *who* or *what* for a direct object of a verb in the embedded clause, to create a *questioned noun phrase* or indirect question: (*I don't know who Jo saw.*). In these clauses, it appears to syntacticians that the *questioned* object of the verb [*VP* saw **who**] has been extracted or moved to a new surface position (following the main clause verb *know*). This is sometimes shown in the phrase-structure diagram by co-indexing an empty *ghost* or trace constituent at the original position of the question pronoun with the question-*NP* appearing at the surface site:

I don't know $[_{NPobj}\,[_{IP}\,[_{NPi}$ *who] Jo saw* $[_{NPi}$ *_]]]*
$[_{NPsubj}[_{IP}$ *Whoever wins the game]] is our hero.*

There are various characteristic types of sentences. Some typical types include:

- Declarative: *I gave her a book.*
- Yes-no question: *Did you give her a book?*
- Wh-question: *What did you give her?*
- Alternatives question: *Did you give her a book, a scarf, or a knife?*
- Tag question: *You gave it to her, didn't you?*
- Passive: *She was given a book.*
- Cleft: *It must have been a book that she got.*
- Exclamative: *Hasn't this been a great birthday!*
- Imperative: *Give me the book.*

2.4.1.3. Parse Tree Representations

Sentences can be diagrammed in parse trees to indicate phrase-internal structure and linear precedence and immediate dominance among phrases. A typical phrase-structure tree for part of an embedded sentence is illustrated in Figure 2.26.

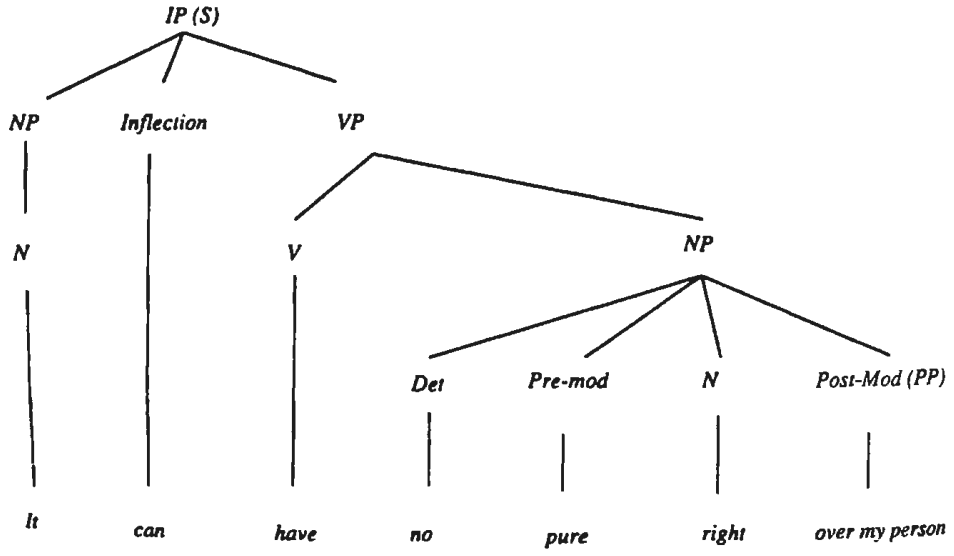


Figure 2.26 A simplified phrase-structure diagram.

For brevity, the same information illustrated in the tree can be represented as a bracketed string as follows:

$[_{IP}\,[_{NP}\,[_{K}\,It\,]_{K}]_{NP}\,[_{I}\,can\,]_{I}\,[_{VP}[_{V}have\,]_{V}\,[_{NP}\,no\ pure\ right\ [_{PP}\,over\ my\ person\,]_{PP}]_{NP}]_{VP}]_{IP}$

With such a bracketed representation, almost every type of syntactic constituent can be coordinated or joined with another of its type, and usually a new phrase node of the common type is added to subsume the constituents such as *NP: We have* $[_{NP}\,[_{NP}$ *tasty berries] and* $[_{NP}$ *tart juices]], IP/S:* $[_{IP}\,[_{IP}$ *Many have come] and* $[_{IP}$ *most have remained]], PP: We went* $[_{PP}\,[_{PP}$ *over the river] and* $[_{PP}$ *into the trees]],* and *VP: We want to* $[_{VP}\,[_{VP}$ *climb the mountains] and* $[_{VP}$ *sail the seas]].*

2.4.2. Semantic Roles

In traditional syntax, grammatical roles are used to describe the direction or control of action relative to the verb in a sentence. Examples include the ideas of *subject, object, indirect object*, etc. Semantic roles, sometimes called case relations, seem similar but dig deeper. They are used to make sense of the participants in an event, and they provide a vocabulary for us to answer the basic question *who did what to whom.* As developed by [13] and others, the theory of semantic roles posits a limited number of universal roles. Each basic meaning of each verb in our mental dictionary is tagged for the obligatory and optional semantic roles used to convey the particular meaning. A typical inventory of case roles is given below:

Agent	*cause or initiator of action, often intentional*
Patient/Theme	*undergoer of the action*
Instrument	*how action is accomplished*
Goal	*to whom action is directed*
Result	*result of action*
Location	*location of action*

These can be realized under various syntactic identities, and can be assigned to both required complement and optional adjuncts. A noun phrase in the Agentive role might be the surface subject of a sentence, or the object of the preposition *by* in a passive. For example, the verb *put* can be considered a process that has, in one of its senses, the case role specifications shown in Table 2.19.

Table 2.19 Analysis of a sentence with *put.*

Analysis	**Example**			
	Kim	*put*	*the book*	*on the table.*
Grammatical functions	*Subject (NP)*	*Predicate (VP)*	*Object (NP)*	*Adverbial (ADVP)*
Semantic roles	*Agent*	*Instrument*	*Theme*	*Location*

Now consider this passive-tense example, where the semantic roles align with different grammatical roles shown in Table 2.20. Words that look and sound identical can have different meaning or different *senses* as shown in Table 2.21. The sporting sense of *put* (as in the sport of shot-put) illustrates the meaning/sense-dependent nature of the role patterns, because in this sense the Locative case is no longer obligatory, as it is in the original sense illustrated in Table 2.19 and Table 2.20.

Table 2.20 Analysis of passive sentence with *put*.

Analysis	**Example**		
	The book	*was put*	*on the table.*
Grammatical functions	*Subject (NP)*	*Predicate (VP)*	*Adverbial (ADVP)*
Semantic roles	*Agent*	*Instrument*	*Location*

Table 2.21 Analysis of a different pattern of *put*.

Analysis	**Example**		
	Kim	*put*	*the shot.*
Grammatical functions	*Subject (NP)*	*Predicate (VP)*	*Object (NP)*
Semantic roles	*Agent*	*Instrument*	*Theme*

The lexical meaning of a verb can be further decomposed into primitive semantic relations such as CAUSE, CHANGE, and BE. The verb *open* might appear as *CAUSE(NP1,PHYSICAL-CHANGE(NP2,NOT-OPEN,OPEN))*. This says that for an agent (*NP1*) to *open* a theme (*NP2*) is to cause the patient to change from a not-opened state to an opened state. Such systems can be arbitrarily detailed and exhaustive, as the application requires.

2.4.3. Lexical Semantics

The specification of particular meaning templates for individual senses of particular words is called *lexical semantics*. When words combine, they may take on propositional meanings resulting from the composition of their meanings in isolation. We could imagine that a speaker starts with a proposition in mind (logical form as will be discussed in the next section), creating a need for particular words to express the idea (lexical semantics); the proposition is then linearized (syntactic form) and spoken (phonological/phonetic form). Lexical semantics is the level of meaning before words are composed into phrases and sentences, and it may heavily influence the possibilities for combination.

Words can be defined in a large number of ways including by relations to other words, in terms of decomposition semantic primitives, and in terms of non-linguistic cognitive constructs, such as perception, action, and emotion. There are hierarchical and non-hierarchical relations. The main hierarchical relations would be familiar to most object-oriented programmers. One is *is-a* taxonomies (a *crow* is-a *bird*), which have transitivity of properties

from type to subtype (inheritance). Another is *has-a* relations (a *car* has-a *windshield*), which are of several differing qualities, including process/subprocess (*teaching* has-a subprocess *giving exams*), and arbitrary or natural subdivisions of part-whole relations (*bread* has-a division into *slices, meter* has-a division into *centimeters*). Then there are non-branching hierarchies (no fancy name) that essentially form scales of degree, such as *frozen* $\Rightarrow$ *cold* $\Rightarrow$ *lukewarm* $\Rightarrow$ *hot* $\Rightarrow$ *burning*. Non-hierarchical relations include synonyms, such as *big/large*, and antonyms such as *good/bad*.

Words seem to have natural affinities and disaffinities in the semantic relations among the concepts they express. Because these affinities could potentially be exploited by future language understanding systems, researchers have used the generalizations above in an attempt to tease out a parsimonious and specific set of basic relations under which to group entire lexicons of words. A comprehensive listing of the families and subtypes of possible semantic relations has been presented in [10]. In Table 2.22, the leftmost column shows names for families of proposed relations, the middle column differentiates subtypes within each family, and the rightmost column provides examples of word pairs that participate in the proposed relation. Note that case roles have been modified for inclusion as a type of semantic relation within the lexicon.

We can see from Table 2.22 that a single word could participate in multiple relations of different kinds. For example, *knife* appears in the examples for *Similars: invited attribute* (i.e., a desired and expected property) as: *knife-sharp*, and also under *Case Relations: action-instrument*, which would label the relation of *knife* to the action *cut* in *He cut the bread with a knife.* This suggests that an entire lexicon could be viewed as a graph of semantic relations, with words or idioms as nodes and connecting edges between them representing semantic relations as listed above. There is a rich tradition of research in this vein.

The biggest practical problem of lexical semantics is the context-dependent resolution of senses of words – so-called polysemy. A classic example is *bank – bank of the stream* as opposed to *money in the bank.* While lexicographers try to identify distinct senses when they write dictionary entries, it has been generally difficult to rigorously quantify exactly what counts as a discrete sense of a word and to disambiguate the senses in practical contexts. Therefore, designers of practical speech understanding systems generally avoid the problem by limiting the domain of discourse. For example, in a financial application, generally only the sense of *bank* as a fiduciary institution is accessible, and others are assumed not to exist. It is sometimes difficult to make a principled argument as to how many distinct senses a word has, because at some level of depth and abstraction, what might appears as separate senses seem to be similar or related, as *face* could be *face of a clock* or *face of person.*

Senses are usually distinguished within a given part-of-speech (POS) category. Thus, when an occurrence of *bank* has been identified as a verb, the *shore* sense might be automatically eliminated, though depending on the sophistication of the system's lexicon and goals, there can be sense differences for many English verbs as well. Within a POS category, often the words that occur near a given ambiguous form in the utterance or discourse are clues to interpretation, where links can be established using semantic relations as described above. Mutual information measures as discussed in Chapter 3 can sometimes provide hints. In a context of dialog where other, less ambiguous financial terms come up

frequently, the sense of *bank* as fiduciary institution is more likely. Finally, when all else fails, often senses can be ranked in terms of their a priori likelihood of occurrence. It should always be borne in mind that language is not static; it can change form under a given analysis at any time. For example, the stable English form *spinster*, a somewhat pejorative term for an older, never-married female, has recently taken on a new morphologically complex form, with the new sense of a high political official, or media spokesperson, employed to provide bland disinformation (*spin*) on a given topic.

Table 2.22 Semantic relations.

Family	Subtype	Example
Contrasts	Contrary	*old-young*
	Contradictory	*alive-dead*
	Reverse	*buy-sell*
	Directional	*front-back*
	Incompatible	*happy-morbid*
	Asymmetric contrary	*hot-cool*
	Attribute similar	*rake-fork*
Similars	Synonymity	*car-auto*
	Dimensional similar	*smile-laugh*
	Necessary attribute	*bachelor-unmarried*
	Invited attribute	*knife-sharp*
	Action subordinate	*talk-lecture*
Class Inclusion	Perceptual subord.	*animal-horse*
	Functional subord.	*furniture-chair*
	State subord.	*disease-polio*
	Activity subord.	*game-chess*
	Geographic subord.	*country-Russia*
	Place	*Germany-Hamburg*
Case Relations	Agent-action	*artist-paint*
	Agent-instrument	*farmer-tractor*
	Agent-object	*baker-bread*
	Action-recipient	*sit-chair*
	Action-instrument	*cut-knife*
Part-Whole	Functional object	*engine-car*
	Collection	*forest-tree*
	Group	*choir-singer*
	Ingredient	*table-wood*
	Functional location	*kitchen-stove*
	Organization	*college-admissions*
	Measure	*mile-yard*

2.4.4. Logical Form

Because of all the lexical, syntactic, and semantic ambiguity in language, some of which requires external context for resolution, it is desirable to have a metalanguage in which to concretely and succinctly express all linguistically possible meanings of an utterance before discourse and world knowledge are applied to choose the most likely interpretation. The favored metalanguage for this purpose is called the predicate logic, used to represent the logical form, or context-independent meaning, of an utterance. The semantic component of many SLU architectures builds on a substrate of two-valued, first-order, logic. To distinguish *shades of meaning* beyond truth and falsity requires more powerful formalisms for knowledge representation.

In a typical first-order system, predicates correspond to events or conditions denoted by verbs (such as *Believe* or *Like*), states of identity (such as being a *Dog* or *Cat*), and properties of varying degrees of permanence (*Happy*). In this form of logical notation, predicates have open places, filled by arguments, as in a programming language subroutine definition. Since individuals may have identical names, subscripting can be used to preserve unique reference. In the simplest systems, predication ranges over individuals rather than higher-order entities such as properties and relations.

Predicates with filled argument slots map onto sets of individuals (constants) in the universe of discourse, in particular those individuals possessing the properties, or participating in the relation, named by the predicate. One-place predicates like *Soldier*, *Happy*, or *Sleeps* range over sets of individuals from the universe of discourse. Two-place predicates, like transitive verbs such as *loves*, range over a set consisting of ordered pairs of individual members (constants) of the universe of discourse. For example, we can consider the universe of discourse to be U = {*Romeo, Juliet, Paris, Rosaline, Tybalt*}, people as characters in a play. They do things with and to one another, such as loving and killing. Then we could imagine the relation *Loves* interpreted as the set of ordered pairs: {<*Romeo, Juliet*>, <*Juliet, Romeo*>, <*Tybalt, Tybalt*>, <*Paris, Juliet*>}, a subset of the Cartesian product of theoretically possible love matches $U \times U$. So, for any ordered pair x, y in U, *Loves*(x, y) is true if the ordered pair <x,y> is a member of the extension of the *Loves* predicate as defined, e.g., *Romeo loves Juliet*, *Juliet loves Romeo*, etc.. Typical formal properties of relations are sometimes specially marked by grammar, such as the reflexive relation *Loves*(*Tybalt*, *Tybalt*), which can rendered in natural language as *Tybalt loves himself*. Not every possibility is present; for instance in our example, the individual Rosaline does not happen to participate at all in this extensional definition of *Loves* over U, as her omission from the pairs list indicates. Notice that the subset of *Loves*(x, y) of ordered pairs involving both *Romeo* and *Juliet* is symmetric, also marked by grammar, as in *Romeo and Juliet love each other*. This general approach extends to predicates with any arbitrary number of arguments, such as intransitive verbs like *give*.

Just as in ordinary propositional logic, connectives such as negation, conjunction, disjunction, and entailment are admitted, and can be used with predicates to denote common natural language meanings:

Romeo isn't happy = ¬Happy(Romeo)
Romeo isn't happy, but Tybalt is (happy) = ¬Happy(Romeo) ∧ Happy(Tybalt)
Either Romeo or Tybalt is happy = Happy(Romeo) ∨ Happy(Tybalt)
If Romeo is happy, Juliet is happy = Happy(Romeo) → Happy(Juliet)

Formulae, such as those above, are also said to bear a binary truth value, true or false, with respect to a world of individuals and relations. The determination of the truth value is compositional, in the sense that the truth value of the whole depends on the truth value of the parts. This is a simplistic but formally tractable view of the relation between language and meaning.

Predicate logic can also be used to denote quantified noun phrases. Consider a simple case such as *Someone killed Tybalt*, predicated over our same U = {*Romeo, Juliet, Paris, Rosaline, Tybalt*}. We can now add an *existential* quantifier, ∃, standing for *there exists* or *there is at least one*. This quantifier will bind a variable over individuals in U, and will attach to a proposition to create a new, quantified proposition in logical form. The use of variables in propositions such as *killed(x, y)* creates open propositions. Binding the variables with a quantifier over them closes the proposition. The quantifier is prefixed to the original proposition: ∃*x Killed(x, Tybalt)*.

To establish a truth (semantic) value for the quantified proposition, we have to satisfy the disjunction of propositions in U: *Killed(Romeo, Tybalt)* ∨ *Killed(Juliet, Tybalt)* ∨ *Killed(Paris, Tybalt)* ∨ *Killed(Rosaline, Tybalt)* ∨ *Killed(Tybalt, Tybalt)*. The set of all such bindings of the variable x is the space that determines the truth or falsity of the proposition. In this case, the binding of x = *Romeo* is sufficient to assign a value true to the existential proposition.

2.5. Historical Perspective and Further Reading

Motivated to improve speech quality over the telephone, AT&T Bell Labs has contributed many influential discoveries in speech hearing, including the critical band and articulation index [2, 3]. The *Auditory Demonstration* CD prepared by Houtsma, Rossing, and Wagenaars [18] has a number of very interesting examples on psychoacoustics and its explanations. *Speech, Language, and Communication* [30] and *Speech Communication – Human and Machine* [32] are two good books that provide modern introductions to the structure of spoken language. Many speech perception experiments were conducted by exploring how phonetic information is distributed in the time or frequency domain. In addition to the formant structures for vowels, frequency importance function [12] has been developed to study how features related to phonetic categories are stored at various frequencies. In the time domain, it has been observed [16, 19, 42] that salient perceptual cues may not be evenly distributed over the speech segments and that certain perceptual critical points exist.

As intimate as speech and acoustic perception may be, there are also strong evidences that lexical and linguistic effects on speech perception are not always consistent with acoustic ones. For instance, it has long been observed that humans exhibit difficulties in distinguishing non-native phonemes. Human subjects also carry out categorical goodness

difference assimilation based on their mother tongue [34], and such perceptual mechanism can be observed as early as in six-month-old infants [22]. On the other hand, hearing-impaired listeners are able to effortlessly overcome their acoustical disabilities for speech perception [8]. Speech perception is not simply an auditory matter. McGurk and MacDonald (1976) [27, 28] dramatically demonstrated this when they created a videotape on which the auditory information (phonemes) did not match the visual speech information. The effect of this mismatch between the auditory signal and the visual signal was to create a third phoneme different from both the original auditory and visual speech signals. An example is dubbing the phoneme */ba/* to the visual speech movements */ga/*. This mismatch results in hearing the phoneme */da/*. Even when subjects know of the effect, they report the McGurk effect percept. The McGurk effect has been demonstrated for consonants. vowels, words, and sentences.

The earliest scientific work on phonology and grammars goes back to Panini, a Sanskrit grammarian of the fifth century B.C. (estimated), who created a comprehensive and scientific theory of phonetics, phonology, and morphology, based on data from Sanskrit (the classical literary language of the ancient Hindus). Panini created formal production rules and definitions to describe Sanskrit grammar, including phenomena such as construction of sentences, compound nouns, etc. Panini's formalisms function as ordered rules operating on underlying structures in a manner analogous to modern linguistic theory. Panini's phonological rules are equivalent in formal power to Backus-Nauer form (BNF). A general introduction to this pioneering scientist is Cardona [9].

An excellent introduction to all aspects of phonetics is *A Course in Phonetics* [24]. A good treatment of the acoustic structure of English speech sounds and a through introduction and comparison of theories of speech perception is to be found in [33]. The basics of phonology as part of linguistic theory are treated in *Understanding Phonology* [17]. An interesting treatment of word structure (morphology) from a computational point of view can be found in *Morphology and Computation* [40]. A comprehensive yet readable treatment of English syntax and grammar can be found in *English Syntax* [4] and *A Comprehensive Grammar of the English Language* [36]. Syntactic theory has traditionally been the heart of linguistics, and has been an exciting and controversial area of research since the 1950s. Be aware that almost any work in this area will adopt and promote a particular viewpoint, often to the exclusion or minimization of others. A reasonable place to begin with syntactic theory is *Syntax: A Minimalist Introduction* [37]. An introductory textbook on syntactic and semantic theory that smoothly introduces computational issues is *Syntactic Theory: A Formal Introduction* [39]. For a philosophical and entertaining overview of various aspects of linguistic theory, see *Rhyme and Reason: An Introduction to Minimalist Syntax* [44]. A good and fairly concise treatment of basic semantics is *Introduction to Natural Language Semantics* [11]. Deeper issues are covered in greater detail and at a more advanced level in *The Handbook of Contemporary Semantic Theory* [25]. The intriguing area of lexical semantics (theory of word meanings) is comprehensively presented in *The Generative Lexicon* [35]. *Concise History of the Language Sciences* [21] is a good edited book if you are interested in the history of linguistics.

REFERENCES

[1] Aliprand, J., *et al.*, *The Unicode Standard, Version 2.0*, 1996, Addison Wesley.

[2] Allen, J.B., "How Do Humans Process and Recognize Speech?," *IEEE Trans. on Speech and Audio Processing*, 1994, **2**(4), pp. 567-577.

[3] Allen, J.B., "Harvey Fletcher 1884–1981" in *The ASA Edition of Speech and Hearing Communication* 1995, Woodbury, New York, pp. A1-A34, Acoustical Society of America.

[4] Baker, C.L., *English Syntax*, 1995, Cambridge, MA, MIT Press.

[5] Blauert, J., *Spatial Hearing*, 1983, MIT Press.

[6] Brill, E., "Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging," *Computational Linguistics*, 1995, **21**(4), pp. 543-566.

[7] Brown, P., *et al.*, "Class-Based N-gram Models of Natural Language," *Computational Linguistics*, 1992, **18**(4).

[8] Caplan, D. and J. Utman, "Selective Acoustic Phonetic Impairment and Lexical Access in an Aphasic Patient," *Journal of the Acoustical Society of America*, 1994, **95**(1), pp. 512-517.

[9] Cardona, G., *Panini: His Work and Its Traditions: Background and Introduction*, 1988, Motilal Banarsidass.

[10] Chaffin, R., and Herrmann, D., "The Nature of Semantic Relations: A Comparison of Two Approaches" in *Representing Knowledge in Semantic Networks*, M. Evens, ed., 1988, Cambridge, UK, Cambridge University Press.

[11] de Swart, H., *Introduction to Natural Language Semantics*, 1998, Stanford, CA, Center for the Study of Language and Information Publications.

[12] Duggirala, V., *et al.*, "Frequency Importance Function for a Feature Recognition Test Material," *Journal of the Acoustical Society of America*, 1988, **83**(9), pp. 2372-2382.

[13] Fillmore, C.J., "The Case for Case" in *Universals in Linguistic Theory*, E. Bach and R. Harms, eds. 1968, New York, NY, Holt, Rinehart and Winston.

[14] Fletcher, H., "Auditory Patterns," *Rev. Mod. Phys.*, 1940, **12**, pp. 47-65.

[15] Fry, D.B., *The Physics of Speech*, Cambridge Textbooks in Linguistics, 1979, Cambridge, U.K., Cambridge University Press.

[16] Furui, S., "On The Role of Spectral Transition for Speech Perception," *Journal of the Acoustical Society of America*, 1986, **80**(4), pp. 1016-1025.

[17] Gussenhoven, C., and Jacobs, H., *Understanding Phonology*, Understanding Language Series, 1998, Edward Arnold.

[18] Houtsma, A., T. Rossing, and W. Wagenaars, *Auditory Demonstrations*, 1987, Institute for Perception Research, Eindhovern, The Netherlands, Acoustic Society of America.

[19] Jenkins, J., W. Strange, and S. Miranda, "Vowel Identification in Mixed-Speaker Silent-Center Syllables," *Journal of the Acoustical Society of America*, 1994, **95**(2), pp. 1030-1041.

[20] Klatt, D., "Review of the ARPA Speech Understanding Project," *Journal of Acoustical Society of America*, 1977, **62**(6), pp. 1324-1366.

[21] Koerner, E. and E. Asher, eds. *Concise History of the Language Sciences*, 1995, Oxford, Elsevier Science.

[22] Kuhl, P., "Infant's Perception and Representation of Speech: Development of a New Theory," *Int. Conf. on Spoken Language Processing*, 1992, Alberta, Canada, pp. 449-452.

[23] Kupeic, J., "Robust Part-of-Speech Tagging Using a Hidden Markov Model," *Computer Speech and Language*, 1992, **6**, pp. 225-242.

[24] Ladefoged, P., *A Course in Phonetics*, 1993, Harcourt Brace Johanovich.

[25] Lappin, S., *The Handbook of Contemporary Semantic Theory*, Blackwell Handbooks in Linguistics, 1997, Oxford, UK, Blackwell Publishsers Inc.

[26] Lindsey, P. and D. Norman, *Human Information Processing*, 1972, New York and London, Academic Press.

[27] MacDonald, J. and H. McGurk, "Visual Influence on Speech Perception Process," *Perception and Psychophysics*, 1978, **24**(3), pp. 253-257.

[28] McGurk, H. and J. MacDonald, "Hearing Lips and Seeing Voices," *Nature*, 1976, **264**, pp. 746-748.

[29] Merialdo, B., "Tagging English Text with a Probabilistic Model," *Computational Linguistics*, 1994, **20**(2), pp. 155-172.

[30] Miller, J. and P. Eimas, *Speech, Language and Communication*, Handbook of Perception and Cognition, eds. E. Carterette and M. Friedman, 1995, Academic Press.

[31] Moore, B.C., *An Introduction to the Psychology of Hearing*, 1982, London, Academic Press.

[32] O'Shaughnessy, D., *Speech Communication – Human and Machine*, 1987, Addison-Wesley.

[33] Pickett, J.M., *The Acoustics of Speech Communication*, 1999, Needham Heights, MA, Allyn & Bacon.

[34] Polka, L., "Linguistic Influences in Adult Perception of Non-native Vowel Contrast," *Journal of the Acoustical Society of America*, 1995, **97**(2), pp. 1286-1296.

[35] Pustejovsky, J., *The Generative Lexicon*, 1998, Bradford Books.

[36] Quirk, R., Svartvik, J., Leech, G., *A Comprehensive Grammar of the English Language*, 1985, Addison-Wesley Pub. Co.

[37] Radford, A., *Syntax: A Minimalist Introduction*, 1997, Cambridge, U.K., Cambridge Univ. Press.

[38] Rossing, T.D., *The Science of Sound*, 1982, Reading, MA, Addison-Wesley.

[39] Sag, I., Wasow, T., *Syntactic Theory: A Formal Introduction*, 1999, Cambridge, UK, Cambridge University Press.

[40] Sproat, R., *Morphology and Computation*, ACL-MIT Press Series in Natural Language Processing, 1992, Cambridge, MA, MIT Press.

[41] Stevens, S.S. and J. Volkman, "The Relation of Pitch to Frequency," *Journal of Psychology*, 1940, **53**, pp. 329.

[42] Strange, W., J. Jenkins, and T. Johnson, "Dynamic Specification of Coarticulated Vowels," *Journal of the Acoustical Society of America*, 1983, 74(3), pp. 695-705.

[43] Thoreau, H.D., *Civil Disobedience, Solitude and Life Without Principle*, 1998, Prometheus Books.

[44] Uriagereka, J., *Rhyme and Reason: An Introduction to Minimalist Syntax*, 1998, Cambridge, MA, MIT Press.

[45] Voutilainen, A., "Morphological Disambiguation" in *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text* 1995, Berlin, Mouton de Gruyter.

[46] Weischedel, R., "BBN: Description of the PLUM System as Used for MUC-6," *The 6th Message Understanding Conferences (MUC-6)*, 1995, San Francisco, Morgan Kaufmann, pp. 55-70.

CHAPTER 12

Basic Search Algorithms

Continuous speech recognition (CSR) is both a pattern recognition and search problem. As described in previous chapters, the acoustic and language models are built upon a statistical pattern recognition framework. In speech recognition, making a search decision is also referred to as decoding. In fact, decoding got its name from information theory (see Chapter 3) where the idea is to *decode* a signal that has presumably been encoded by the source process and has been transmitted through the communication channel, as depicted in Chapter 1, Figure 1.1. In this chapter, we first review the general decoder architecture that is based on such a source-channel model.

The decoding process of a speech recognizer is to find a sequence of words whose corresponding acoustic and language models best match the input signal. Therefore, the process of such a decoding process with trained acoustic and language models is often referred to as just a *search* process. Graph search algorithms have been explored extensively in the fields of artificial intelligence, operation research, and game theory. In this chapter first we present several basic search algorithms, which serve as the basic foundation for CSR.

The complexity of a search algorithm is highly correlated with the search space, which is determined by the constraints imposed by the language models. We discuss the impact of different language models, including finite-state grammars, context-free grammars, and n-grams.

Speech recognition search is usually done with the Viterbi or A* stack decoders. The reasons for choosing the Viterbi decoder involve arguments that point to speech as a left-to-right process and to the efficiencies afforded by a time-synchronous process. The reasons for choosing a stack decoder involve its ability to more effectively exploit the A* criteria, which holds out the hope of performing an optimal search as well as the ability to handle huge search spaces. Both algorithms have been successfully applied to various speech recognition systems. The relative merits of both search algorithms were quite controversial in the 1980s. Lately, with the help of efficient pruning techniques, Viterbi beam search has been the preferred method for almost all speech recognition tasks. Stack decoding, on the other hand, remains an important strategy to uncover the n-best and lattice structures.

12.1. BASIC SEARCH ALGORITHMS

Search is a subject of interest in artificial intelligence and has been well studied for expert systems, game playing, and information retrieval. We discuss several general graph search methods that are fundamental to spoken language systems. Although the basic concept of graph search algorithms is independent of any specific task, the efficiency often depends on how we exploit domain-specific knowledge.

The idea of search implies moving around, examining things, and making decisions about whether the sought object has yet been found. In general, search problems can be represented using the *state-space search* paradigm. It is defined by a triplet (S, O, G), where S is a set of initial states, O a set of operators (or rules) applied on a state to generate a transition with its corresponding cost to another state, and G a set of goal states. A solution in the state-space search paradigm consists in finding a path from an initial state to a goal state. The state-space representation is commonly identified with a directed graph in which each node corresponds to a state and each arc to an application of an operator (or a rule), which transitions from one state to another. Thus, the state-space search is equivalent to searching through the graph with some objective function.

Before we present any graph search algorithms, we need to remind the readers of the importance of the dynamic programming algorithm described in Chapter 8. Dynamic programming should be applied whenever possible and as early as possible because (1) unlike any heuristics, it will not sacrifice optimality; (2) it can transform an exponential search into a polynomial search.

12.1.1. General Graph Searching Procedures

Although dynamic programming is a powerful polynomial search algorithm, many interesting problems cannot be handled by it. A classical example is the traveling salesman's problem. We need to find a shortest-distance tour, starting at one of many cities, visiting each city exactly once, and returning to the starting city. This is one of the most famous problems in the *NP*-hard class [1, 32]. Another classical example is the *N*-queens problem (typically 8-queens), where the goal is to place N queens on an $N \times N$ chessboard in such a way that no queen can capture any other queen, i.e., there is no more than one queen in any given row, column, or diagonal. Many of these puzzles have the same characteristics. As we know, the best algorithms currently known for solving the *NP*-hard problem are exponential in the problem size. Most graph search algorithms try to solve those problems using heuristics to avoid or moderate such a combinatorial explosion.

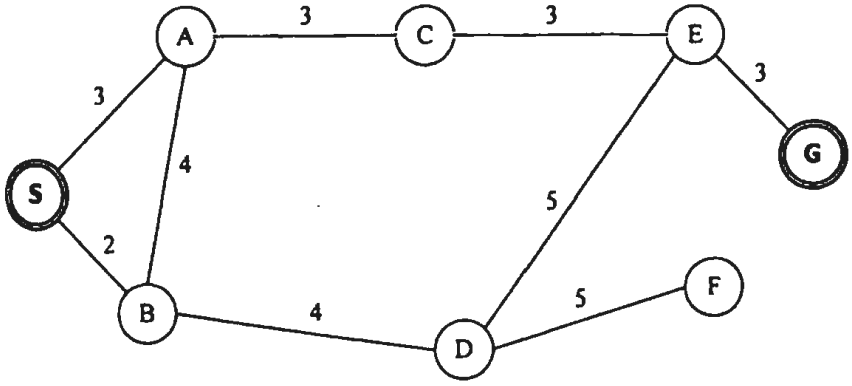


Figure 12.1 A highway distance map for cities S, A, B, C, D, E, F, and G. The salesman needs to find a path to travel from city S to city G [42].

Let's start our discussion of graph search procedure with a simple city-traveling problem [42]. Figure 12.1 shows a highway distance map for all the cities. A salesman named John needs to travel from the starting city S to the end city G. One obvious way to find a path is to derive a graph that allows orderly exploration of all possible paths. Figure 12.2 shows the graph that traces out all possible paths in the city-distance map shown in Figure 12.1. Although the city-city connection is bi-directional, we should note that the search graph in this case must not contain cyclic paths, because they would not lead to any progress in this scenario.

If we define the search space as the potential number of nodes (states) in the graph search procedure, the search space for finding the optimal state sequence in the Viterbi algorithm (described in Chapter 8) is $N \times T$, where N is the number of states for the HMM and T is the length of the observation. Similary, the search space for John's traveling problem will be 27.

Another important measure for a search graph is the *branching factor*, defined as the average number of successors for each node. Since the number of nodes of a search graph

(or tree) grows exponentially with base equal to this branching factor, we certainly need to watch out for search graphs (or trees) with a large branching factor. Sometimes they can be too big to handle (even infinite, as in game playing). We often trade the optimal solution for improved performance and feasibility. That is, the goal for such search problems is to find one satisfactory solution instead of the optimal one. In fact, most AI (artifical intelligence) search problems belong to this category.

The search tree in Figure 12.2 may be implemented either explicitly or implicitly. In an explicit implementation, the nodes and arcs with their corresponding distances (or costs) are explicitly specified by a table. However, an explicit implementation is clearly impractical for large search graphs and impossible for those with infinite nodes. In practice, most parts of the graph may never be explored before a solution is found. Therefore, a sensible strategy is to dynamically generate the search graph. The part that becomes explicit is often referred to as an *active* search space. Throughout the discussion here, it is important to keep in mind this distinction between the implicit search graph that is specified by the start node S and the explicit partial search graphs that are actually constructed by the search algorithm.

To expand the tree, the term *successor operator* (or *move generator*, as it is often called in game search) is defined as an operator that is applied to a node to generate all of the successors of that node and to compute the distance associated with each arc. The successor operator obviously depends on the topology (or rules) of the problem space. Expanding the starting node S, and successors of S, ad infinitum, gradually makes the implicitly

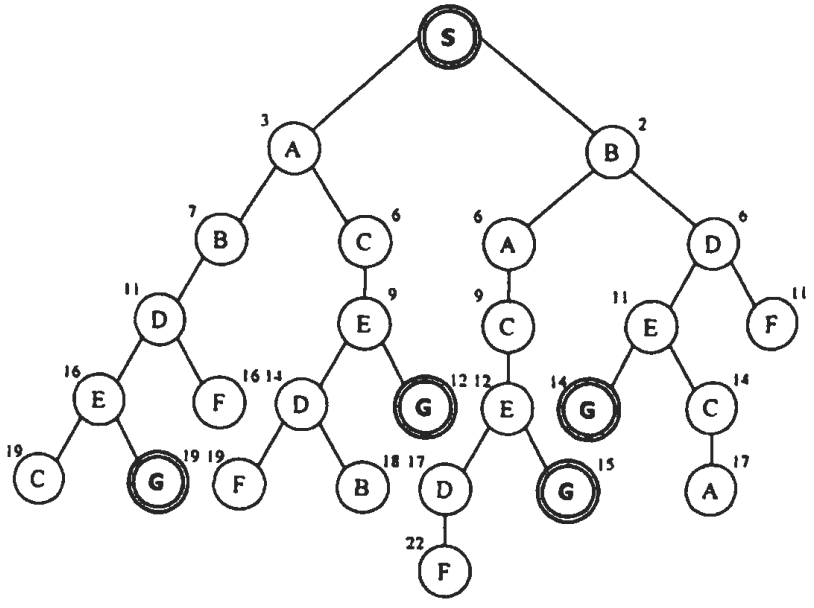


Figure 12.2 The search tree (graph) for the salesman problem illustrated in Figure 12.1. The number next to each node is the accumulated distance from start city to end city [42].

defined graph explicit. This recursive procedure is straightforward, and the search graph (tree) can be constructed without the extra bookkeeping. However, this process would only generate a search tree where the same node might be generated as a part of several possible paths.

For example, node E is being generated in four different paths. If we are interested in finding an optimal path to travel from S to G, it is more efficient to merge those different paths that lead to the same node E. We can pick the shortest path up to C, since everything following E is the same for the rest of the paths. This is consistent with the dynamic programming principle—when looking for the best path from S to G, all partial paths from S to any node E, other than the best path from S to E, should be discarded. The dynamic programming merge also eliminates cyclic paths implicitly, since a cyclic path cannot be the shortest path. Performing this extra bookkeeping (merging different paths leading into the same node) generates a search graph rather than a search tree.

Although a graph search has the potential advantage over a tree search of being more efficient, it does require extra bookkeeping. Whether this effort is justified depends on the individual problem one has to address.

Most search strategies search in a forward direction, i.e., build the search graph (or tree) by starting with the initial configuration (the starting state S) from the root. In the general AI literature, this is referred to as *forward reasoning* [43], because it performs rule-base reasoning by matching the left side of rules first. However, for some specific problem domains, it might be more efficient to use *backward reasoning* [43], where the search graph is built from the bottom up (the goal state G). Possible scenarios include:

- *There are more initial states than goal states.* Obviously it is easy to start with a small set of states and search for paths leading to one of the bigger sets of states. For example, suppose the initial state S is the hometown for John in the city-traveling problem in Figure 12.1 and the goal state G is an unfamiliar city for him. In the absence of a map, there are certainly more locations (neighboring cities) that John can identify as being close[1] to his home city S than those he can identify as being close to an unfamiliar location. In a sense, all of those locations being identified as close to John's home city S are equivalent to the initial state S. This means John might want to consider reasoning backward from the unfamiliar goal city G for the trip planning.
- *The branching factor for backward reasoning is smaller than that for forward reasoning.* In this case it makes sense to search in the direction with lower branching factor.

It is in principle possible to search from both ends simultaneously, until two partial paths meet somewhere in the middle. This strategy is called *bi-directional search* [43]. Bi-directional search seems particularly appealing if the number of nodes at each step grows

[1] *Being close* means that, once John reaches one of those neighboring cities, he can easily remember the best path to return home. It is similar to the killer book for chess play. Once the player reaches a particular board configuration, he can follow the killer book for moves that can guarantee a victory.

exponentially with the depth that needs to be explored. However, sometimes bi-directional search can be devastating. The two searches may cross each other, as illustrated in Figure 12.3.

The process of explicitly generating part of an implicitly defined graph forms the essence of our general graph search procedure. The procedure is summarized in Algorithm 12.1. It maintains two lists: *OPEN*, which stores the nodes waiting for expansion, and *CLOSE*, which stores the already expanded nodes. Steps 6a and 6b are basically the bookkeeping process to merge different paths going into the same node by picking the one that has the minimum distance. Step 6a handles the case where v is in the *OPEN* list and thus is not expanded. The merging process is straightforward, with a single comparison and change of traceback pointer if necessary. However, when v is in the *CLOSE* list and thus is already expanded in Step 6b, the merging requires additional forward propagation of the new score if the current path is found to be better than the best subpath already in the *CLOSE* list. This forward propagation could be very expensive. Fortunately, most of the search strategy can avoid such a procedure if we know that the already expanded node must belong in the best path leading to it. We discuss this in Section 12.5.

As described earlier, it may not be worthwhile to perform bookkeeping for a graph search, so Steps 6a and 6b are optional. If both steps are omitted, the graph search algorithm described above becomes a tree search algorithm. To illustrate different search strategies, tree search is used as the basic graph search algorithm in the sections that follows. However, you should note that all the search methods described here could be easily extended to graph search with the extra bookkeeping (merging) process as illustrated in Steps 6a and 6b of Algorithm 12.1.

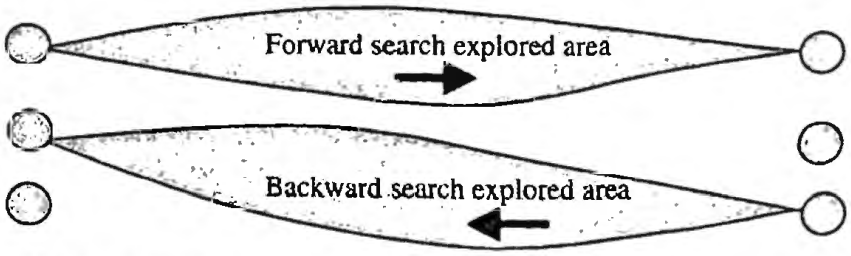


Figure 12.3 A bad case for bi-directional search, where the forward search and the backward search crossed each other [42].

ALGORITHM 12.1: *THE GRAPH-SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty *CLOSE* list
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: Pop up the first node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
Step 5: Expand node N by applying the successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the ancestors of N from $SS(N)$.
Step 6: $\forall v \in SS(N)$ do

6a. (optional) If $v \in OPEN$ and the accumulated distance of the new path is smaller than that for the one in the *OPEN* list, do

(i) change the traceback (parent) pointer of v to N and adjust the accumulated distance for v.

(ii) go to Step 7.

6b. (optional) If $v \in CLOSE$ and the accumulated distance of the new path is smaller than the partial path ending at v in the *CLOSE* list, do

(i) change the traceback (parent) pointer of v to N and adjust the accumulated distance for all paths that contain v.

(ii) go to Step 7.

6c. Create a pointer pointing to N and push it into the *OPEN* list.
Step 7: Reorder the *OPEN* list according to search strategy or some heuristic measurement.
Step 8: Go to Step 2.

12.1.2. Blind Graph Search Algorithms

If the aim of the search problem is to find an acceptable path instead of the best path, blind search is often used. *Blind search* treats every node in the *OPEN* list the same and blindly decides the order to be expanded without using any domain knowledge. Since blind search treats every node equally, it is often referred to as *uniform search* or *exhaustive search*, because it exhaustively tries out all possible paths. In AI, people are typically not interested in blind search. However, it does provide a lot of insight into many sophisticated heuristic search algorithms. You should note that blind search does not expand nodes randomly. Instead, it follows some systematic way to explore the search graph. Two popular types of blind search are depth-first search and breadth-first search.

12.1.2.1. Depth-First Search

When we are in a maze, the most natural way to find a way out is to mark the branch we take whenever we reach a branching point. The marks allow us to go back to a choice point with an unexplored alternative, withdraw the most recently made choice and undo all consequences of the withdrawn choice whenever a dead-end is reached. Once the alternative choice is selected and marked, we go forward based on the same procedure. This intuitive search strategy is called *backtracking*. The famous *N*-queens puzzle [32] can be handily solved by the backtracking strategy.

Depth-first search picks an arbitrary alternative at every node visited. The search sticks with this partial path and works forward from the partial path. Other alternatives at the same level are ignored completely (for the time being) in the hope of finding a solution based on the current choice. This strategy is equivalent to ordering the nodes in the *OPEN* list by their depth in the search graph (tree). The deepest nodes are expanded first and nodes of equal depth are ordered arbitrarily.

Although depth-first search hopes the current choice leads to a solution, sometimes the current choice could lead to a dead-end (a node which is neither a goal node nor can be expanded further). In fact, it is desirable to have many short dead-ends. Otherwise, the algorithm may search for a very long time before it reaches a dead-end, or it might not ever reach a solution if the search space is infinite. When the search reaches a dead-end, it goes back to the last decision point and proceeds with another alternative.

Figure 12.4 shows all the nodes being expanded under the depth-first search algorithm for the city-traveling problem illustrated in Figure 12.1. The only differences between the graph search and the depth-first search algorithms are:

1. The graph search algorithm generates all successors at a time (although all except one are ignored first), while depth-first search generates only one successor at a time.
2. The graph search, when successfully finding a path, saves only one path from the starting node to the goal node, while depth-first search in general saves the entire record of the search graph.

Depth-first search could be dangerous because it might search an impossible path that is actually an infinite dead-end. To prevent exploring of paths that are too long, a depth bound can be placed to constrain the nodes to be expanded, and any node reaching that depth limit is treated as a terminal node (as if it had no successor).

The general graph search algorithm can be modified into a depth-first search algorithm as illustrated in Algorithm 12.2.

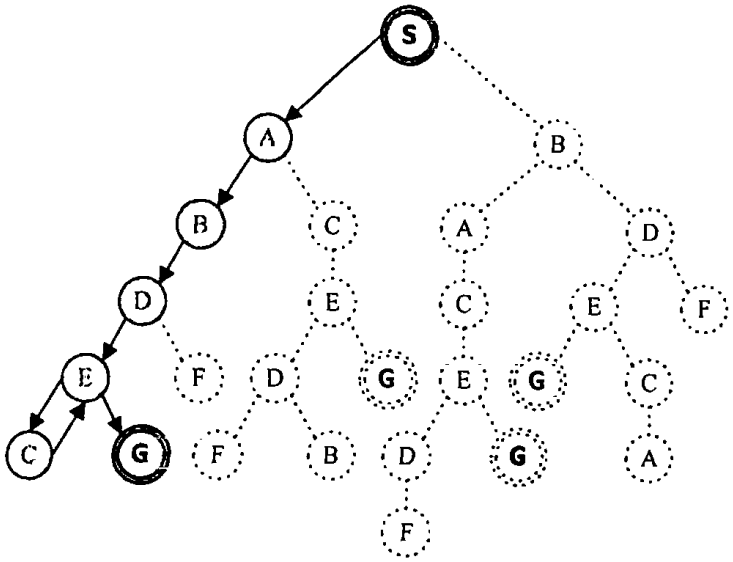


Figure 12.4 The node-expanding procedure of the depth-first search for the path search problem in Figure 12.1. When it fails to find the goal city in node *C*, it backtracks to the parent and continues the search until it finds the goal city. The gray nodes are those that are explored. The dotted nodes are not visited during the search [42].

ALGORITHM 12.2: *THE DEPTH-FIRST SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty the CLOSE list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: Pop up the first node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
4a. If the depth of node N is equal to the depth bound, go to Step 2.
Step 5: Expand node N by applying the successor operator to generate the successor set SS(N) of node N. Be sure to eliminate the ancestors of N from SS(N).
Step 6: $\forall v \in SS(N)$ do
6c. Create a pointer pointing to N and push it into the *OPEN* list.
Step 7: Reorder the the *OPEN* list in descending order of the depth of the nodes.
Step 8: Go to Step 2.

12.1.2.2. Breadth-First Search

One natural alternative to the depth-first search strategy is breadth-first search. *Breadth-first search* examines all the nodes on one level before considering any of the nodes on the next level (depth). As shown in Figure 12.5, node *B* would be examined just after node *A*. The search moves on level-by-level, finally discovering G on the fourth level.

Breadth-first search is guaranteed to find a solution if one exists, assuming that a finite number of successors (branches) always follow any node. The proof is straightforward. If there is a solution, its path length must be finite. Let's assume the length of the solution is *M*. Breadth-first search explores all paths of the same length increasingly. Since the number of paths of fixed length *N* is always finite, it eventually explores all paths of length *M*. By that time it should find the solution.

It is also easy to show that a breadth-first search can work on a search tree (graph) with infinite depth on which an unconstrained depth-first search will fail. Although a breadth-first might not find a shortest-distance path for the city-travel problem, it is guaranteed to find the one with fewest cities visited (minimum-length path). In some cases, it is a very desirable solution. On the other hand, a breadth-first search may be highly inefficient when all solutions leading to the goal node are at approximately the same depth. The breadth-first search algorithm is summarized in Algorithm 12.3.

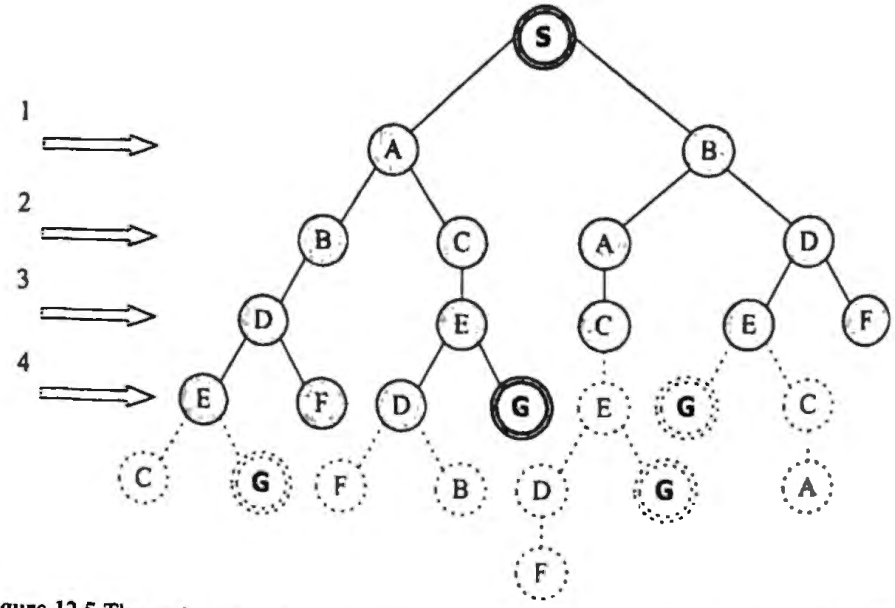


Figure 12.5 The node-expanding procedure of a breadth-first search for the path search problem in Figure 12.1. It searches through each level until the goal is identified. The gray nodes are those that are explored. The dotted nodes are not visited during the search [42].

ALGORITHM 12.3: *THE BREADTH-FIRST SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty the *CLOSE* list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: Pop up the first node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
Step 5: Expand node N by applying the successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the ancestors of N, from $SS(N)$.
Step 6: $\forall v \in SS(N)$ do

6c. Create a pointer pointing to N and push it into the *OPEN* list.

Step 7: Reorder the *OPEN* list in increasing order of the depth of the nodes.
Step 8. Go to Step 2.

12.1.3. Heuristic Graph Search

Blind search methods, like depth-first search and breadth-first search, have no sense (or guidance) of where the goal node lies ahead. Consequently, they often spend a lot of time searching in hopeless directions. If there is guidance, the search can move in the direction that is more likely to lead to the goal. For example, you may want to find a driving route to the World Trade Center in New York. Without a map at hand, you can still use a straight-line distance estimated by eye as a hint to see if you are closer to the goal (World Trade Center). This *hill-climbing* style of guidance can help you to find the destination much more efficiently.

Blind search finds only one arbitrary solution instead of the optimal solution. To find the optimal solution with depth-first or breadth-first search, you must not stop searching when the first solution is discovered. Instead, the search needs to continue until it reaches all the solutions, so you can compare them to pick the best. This strategy for finding the optimal solution is called *British Museum search* or *brute-force search*. Obviously, it is unfeasible when the search space is large. Again, to conduct selective search and yet still be able to find the optimal solution, some guidance on the search graph is necessary.

The guidance obviously comes from domain-specific knowledge. Such knowledge is usually referred to as *heuristic* information, and search methods taking advantage of it are called *heuristic search* methods. There is usually a wide variety of different heuristics for the problem domain. Some heuristics can reduce search effort without sacrificing optimality, while other can greatly reduce search effort but provide only sub-optimal solutions. In most practical problems, the choice of different heuristics is usually a tradeoff between the quality of the solution and the cost of finding the solution.

Heuristic information works like an evaluation function $h(N)$ that maps each node N to a real number, and which serves to indicate the relative goodness (or cost) of continuing the search path from that node. Since in our city-travel problem, straight-line distance is a natural way of measuring the goodness of a path, we can use the heuristic function $h(N)$ for the distance evaluation as:

$h(N)$=Heuristic estimate of the remaining distance from node N to goal G (12.1)

Since $g(N)$, the distance of the partial path to the current node N, is generally known, we have:

$g(N)$=The distance of the partial path already traveled from root S to node N (12.2)

We can define a new heuristic function, $f(N)$, which estimates the total distance for the path (not yet finished) going through node N.

$$f(N) = g(N) + h(N) \tag{12.3}$$

A heuristic search method basically uses the heuristic function $f(N)$ to re-order the *OPEN* list in the Step 7 of Algorithm 12.1. The node with the best heuristic value is explored first (expanded first). Some heuristic search strategies also prune some unpromising partial paths forever to save search space. This is why heuristic search is often referred to as heuristic pruning.

The choice of the heuristic function is critical to the search results. If we use one that overestimates the distance of some nodes, the search results may be suboptimal. Therefore, heuristic functions that do not overestimate the distance are often used in search methods aiming to find the optimal solution.

To close this section, we describe two of the most popular heuristic search methods: best-first (or A* Search) [32, 43] and beam search [43]. They are widely used in many components of spoken language systems.

12.1.3.1. Best-First (A* Search)

Once we have a reasonable heuristic function to evaluate the goodness of each node in the *OPEN* list, we can explore the best node (the node with smallest $f(N)$ value) first, since it offers the best hope of leading to the best path. This natural search strategy is called *best-first search*. To implement best-first search based on the Algorithm 12.1, we need to first evaluate $f(N)$ for each successor before putting the successors in the *OPEN* list in Step 6. We also need to sort the elements in the *OPEN* list based on $f(N)$ in Step 7, so that the best node is in the front-most position waiting to be expanded in Step 3. The modified procedure for performing best-first search is illustrated in Algorithm 12.4. To avoid duplicating nodes in the *OPEN* list, we include Steps 6a and 6b to take advantage of the dynamic programming principle. They perform the needed bookkeeping process to merge different paths leading into the same node.

ALGORITHM 12.4: *THE BEST-FIRST SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty the *CLOSE* list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3. Pop up the first node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
Step 4: If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
Step 5: Expand node N by applying the successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the ancestors of N, from $SS(N)$.
Step 6: $\forall v \in SS(N)$ do

6a. (optional) If $v \in OPEN$ and the accumulated distance of the new path is smaller than that for the one in the the *OPEN* list, do

(i) Change the traceback (parent) pointer of v to N and adjust the accumulated distance for v.

(ii) Evaluate heuristic function $f(v)$ for v and go to Step 7.

6b. (optional) If $v \in CLOSE$ and the accumulated distance of the new path is small than the partial path ending at v in the the *CLOSE* list,

(i) Change the traceback (parent) pointer of v to N and adjust the accumulated distance and heuristic function f for all the paths containing v.

(ii) go to Step 7.

6c. Create a pointer pointing to N and push it into the *OPEN* list.
Step 7: Reorder the the *OPEN* list in the increasing order of the heuristic function $f(N)$.
Step 8: Go to Step 2.

A search algorithm is said to be *admissible* if it can guarantee to find an optimal solution, if one exists. Now we show that if the heuristic function $h(N)$ of estimating the remaining distance from N to goal node G is an underestimate[2] of the true distance from N to goal node G, the best-first search illustrated in Algorithm 12.4 is admissible. In fact, when $h(N)$ satisfies the above criterion, the best-first algorithm is called A* (pronounced as *leh*l-star) Search.

The proof can be carried out informally as follows. When the frontmost node in the *OPEN* list is the goal node G in Step 4, it immediately implies that

$$\forall v \in OPEN \quad f(v) \geq f(G) = g(G) + h(G) = g(G) \tag{12.4}$$

[2] For admissibility, we actually require only that the heuristic function not overestimate the distance from N to G. Since it is very rare to have an exact estimate, we use underestimate throughout this chapter without loss of generality. Sometimes we refer to an underestimate function as a lower-bound estimate of the true value.

Equation (12.4) says that the distance estimate of any incomplete path is no shorter than the first found complete path. Since the distance estimate for any incomplete path is underestimated, the first found complete path in Step 4 must be the optimal path. A similar argument can also be used to prove that the Step 6b is actually not necessary for admissible heuristic functions; that is, there cannot be another path with a shorter distance from the starting node to a node that has been expanded. This is a very important feature since Step 6b is, in general, very expensive and it requires significant updates of many already expanded paths.

The A* search method is actually a family of search algorithms. When $h(N) = 0$ for all N, the search degenerates into an uninformed search[3] [40]. In fact, this type of uninformed search is the famous *branch-and-bound search* algorithm that is often used in many *operations research* problems. Branch-and-bound search always expands the shortest path leading into an open node until there is a path reaching the goal that is of a length no longer than all incomplete paths terminating at open nodes. When $g(N)$ is defined as the depth of the node N, the use of heuristic function $f(N)$ makes the search method identical to breadth-first search. In Section 12.1.2.2, we mention that breadth-first search is guaranteed to find a minimum length path. This can certainly be derived from the admissibility of the A* search method.

When the heuristic function is close to the true remaining distance, the search can usually find the optimal solution without too much effort. In fact, when the true remaining distances for all nodes are known, the search can be done in a totally greedy fashion without any search at all, i.e., the only path explored is the solution. Any non-zero heuristic function is then called an informed heuristic function, and the search using such a function is called informed search. A heuristic function h_1 is said to be more informed than a heuristic function h_2 if the estimate h_1 is everywhere larger than h_2 and yet still admissible (underestimate). Finding an informed admissible heuristic function (guaranteed to underestimate for all nodes) is, in general, a difficult task. The heuristic often requires extensive analysis of the domain-specific knowledge and knowledge representation.

Let's look at a simple example—the 8-puzzle problem. The 8-puzzle consists of eight numbered, movable tiles set in a 3×3 frame. One cell of this frame is always empty, so it is possible to move an adjacent numbered tile into the empty cell. A solution for the 8-puzzle is to find a sequence of moves to change the initial configuration into a given goal configuration as shown in Figure 12.6. One choice for an informed admissible heuristic function h_1 is the number of misplaced tiles associated with the current configuration. Since each misplaced tile needs to move at least once to be in the right position, this heuristic function is clearly a lower bound of the true movements remaining. Based on this heuristic function, the value for the initial configuration will be 7 in Figure 12.7. If we examine this problem further, a more informed heuristic function h_2 can be defined as the sum of all row and column distances of all misplaced tiles and their goal positions. For example, the row and column distance between the tile 8 in the initial configuration and the goal position is 2 + 1= 3,

[3] In some literature an uninformed search is referred to as uniform-cost search.

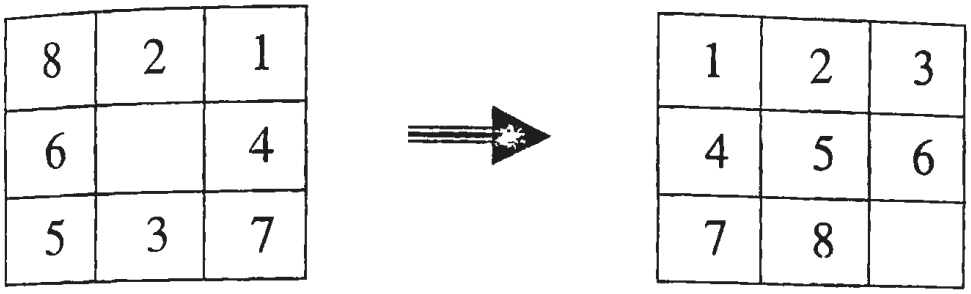


Figure 12.6 Initial and goal configurations for the 8-puzzle problem.

which indicates that one must move tile 8 at least 3 times in order for it to be in the right position. Based on the heuristic function h_2, the value for the initial configuration will be 16 in Figure 12.6. h_2 is again admissible.

In our city-travel problem, one natural choice for the underestimating heuristic function of the remaining distance between node N and goal G is the straight-line distance since the true distance must be no shorter than the straight-line distance.

Figure 12.7 shows an augmented city-distance map with straight-line distance to goal node attached to each node. Accordingly, the heuristic search tree can be easily constructed for improved efficiency. Figure 12.8 shows the search progress of applying the A* search algorithm for the city-traveling problem by using the straight-line distance heuristic function to estimate the remaining distances.

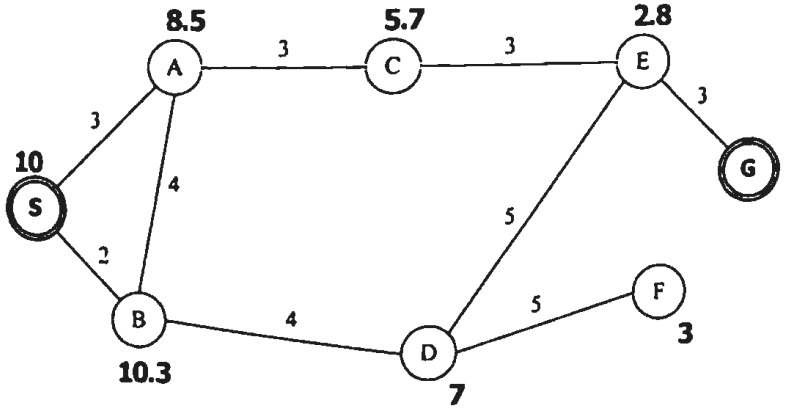


Figure 12.7 The city-travel problem augmented with heuristic information. The numbers beside each node indicate the straight-line distance to the goal node G [42].

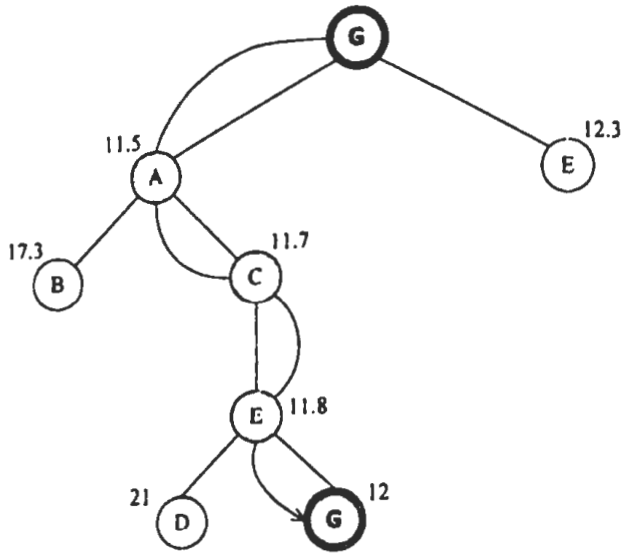


Figure 12.8 The search progress of applying A* search for the city-travel problem. The search determines that path S-A-C-E-G is the optimal one. The number beside the node is f values on which the sorting of the *OPEN* list is based [42].

12.1.3.2. Beam Search

Sometimes, it is impossible to find any effective heuristic estimate, as required in A* search, particularly when there is very little (or no) information about the remaining paths. For example, in real-time speech recognition, there is little information about what the speaker will utter for the remaining speech. Therefore, an efficient uninformed search strategy is very important to tackle this type of problem.

Breadth-first style search is an important strategy for heuristic search. A breadth-first search virtually explores all the paths with the same depth before exploring deeper paths. In practice, paths of the same depth are often easier to compare. It requires fewer heuristics to rank the goodness of each path. Even with uninformed heuristic function ($h(N)=0$), the direct comparison of g (distance so far) of the paths with the same length should be a reasonable choice.

Beam search is a widely used search technique for speech recognition systems [26, 31, 37]. It is a breadth-first style search and progresses along with the depth. Unlike traditional breadth-first search, however, beam search only expands nodes that are likely to succeed at each level. Only these nodes are kept in the beam, and the rest are ignored (pruned) for improved efficiency.

In general, a beam search only keeps up to w best paths at each stage (level), and the rest of the paths are discarded. The number w is often referred to as beam width. The number of nodes explored remains manageable in beam search even if the whole search space is gigantic. If a beam width w is used in a beam search with an average branching factor b, only $w \times b$ nodes need to be explored at any depth, instead of the exponential number

needed for breadth-first search. Suppose that a beam width of 2 is used for the city-travel problem. Figure 12.9 illustrates how beam search progresses to find the path. We can also see that the beam search saved a large number of unneeded nodes, as shown by the dotted nodes.

The beam search algorithm can be easily modified from the breadth-first search algorithm and is illustrated in Algorithm 12.5. For simplicity, we do not include the merging step here. In Algorithm 12.5, Step 4 obviously requires sorting, which is time-consuming if the number $w \times b$ is huge. In practice, the beam is usually implemented as a flexible list where nodes are expanded if their heuristic functions $f(N)$ are within some threshold (a.k.a., beam threshold) of the best node (the smallest value) at the same level. Thus, we only need to identify the best node and then prune away nodes that are outside of the threshold. Although this makes the beam size change dynamically, it significantly reduces the effort for sorting of the *Beam-Candidate* list. In fact, by adjusting the beam threshold, the beam size can be controlled indirectly and yet kept manageable.

Unlike A' search, beam search is an approximate heuristic search method that is not admissible. However, it has a number of unique merits. Because of its simplicity in both its search strategy and its requirement of domain-specific heuristic information, it has become one of the most popular methods for complicated speech recognition problems. It is particularly attractive when integration of different knowledge sources is required in a time-synchronous fashion. It has the advantages of providing a consistent way of exploring nodes level by level and of offering minimally needed communication between different paths. It is also very suitable for parallel implementation because of its breadth-first search nature.

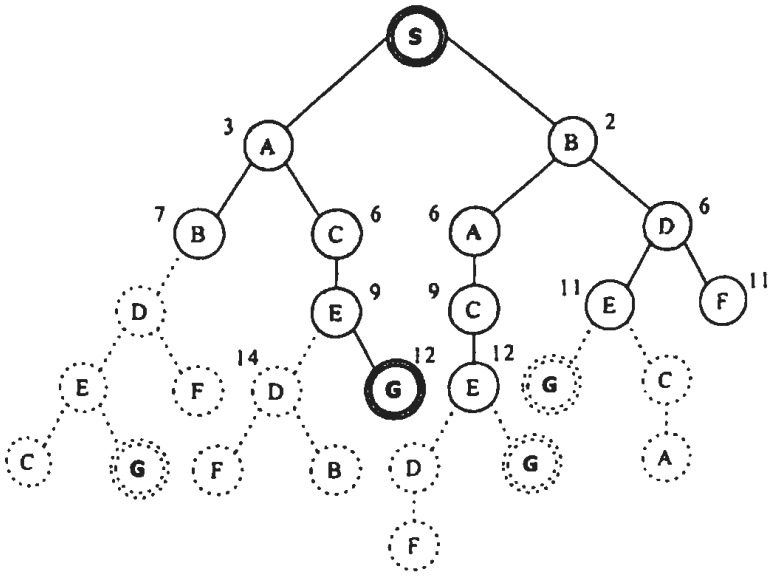


Figure 12.9 Beam search for the city-travel problem. The nodes with gray color are the ones kept in the beam. The transparent nodes were explored but pruned because of higher cost. The dotted nodes indicate all the savings because of pruning [42].

ALGORITHM 12.5: *THE BEAM SEARCH ALGORITHM*

Step 1: Initialization: Put S in the *OPEN* list and create an initially empty *CLOSE* list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: $\forall N \in OPEN$ do
3a. Pop up node N in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
3b. If node N is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from N to S.
3c. Expand node N by applying a successor operator to generate the successor set $SS(N)$ of node N. Be sure to eliminate the successors, which are ancestors of N, from $SS(N)$.
3d. $\forall v \in SS(N)$ Create a pointer pointing to N and push it into *Beam-Candidate* list.
Step 4: Sort the *Beam-Candidate* list according to the heuristic function $f(N)$ so that the best w nodes can be pushed into the the *OPEN* list. Prune the rest of nodes in the *Beam-Candidate* list.
Step 5: Go to Step 2.

12.2. SEARCH ALGORITHMS FOR SPEECH RECOGNITION

As described in Chapter 9, the decoder is basically a search process to uncover the word sequence $\hat{\mathbf{W}} = w_1 w_2 ... w_m$ that has the maximum posterior probability $P(\mathbf{W}|\mathbf{X})$ for the given acoustic observation $\mathbf{X} = X_1 X_2 ... X_n$. That is,

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) = \arg\max_{\mathbf{W}} \frac{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}{P(\mathbf{X})} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X}|\mathbf{W}) \qquad (12.5)$$

One obvious way is to search all possible word sequences and select the one with the best posterior probability score.

The unit of acoustic model $P(\mathbf{X}|\mathbf{W})$ is not necessary a word model. For large-vocabulary speech recognition systems, subword models, which include phonemes, demisyllables, and syllables are often used. When subword models are used, the word model $P(\mathbf{X}|\mathbf{W})$ is then obtained by concatenating the subword models according to the pronunciation transcription of the words in a lexicon or dictionary.

When word models are available, speech recognition becomes a search problem. The goal for speech recognition is thus to find a sequence of word models that best describes the input waveform against the word models. As neither the number of words nor the boundary of each word or phoneme in the input waveform is known, appropriate search strategies to deal with these variable-length nonstationary patterns are extremely important.

When HMMs are used for speech recognition systems, the states in the HMM can be expanded to form the state-search space in the search. In this chapter, we use HMMs as our speech models. Although the HMM framework is used to describe the search algorithms, all

techniques mentioned in this and the following chapter can be used for systems based on other modeling techniques, including template matching and neural networks. In fact, many search techniques had been invented before HMMs were applied to speech recognition. Moreover, the HMMs state transition network is actually general enough to represent the general search framework for all modeling approaches.

12.2.1. Decoder Basics

The lessons learned from dynamic programming or the Viterbi algorithm introduced in Chapter 8 tell us that the exponential blind search can be avoided if we can store some intermediate optimal paths (results). Those intermediate paths are used for other paths without being recomputed each time. Moreover, the beam search described in the previous section shows us that efficient search is possible if appropriate pruning is employed to discard highly unlikely paths. In fact, all the search techniques use two strategies: sharing and pruning. *Sharing* means that intermediate results can be kept, so that they can be used by other paths without redundant re-computation. *Pruning* means that unpromising paths can be discarded reliably without wasting time in exploring them further.

Search strategies based on dynamic programming or the Viterbi algorithm with the help of clever pruning, have been applied successfully to a wide range of speech recognition tasks [31], ranging from small-vocabulary tasks, like digit recognition, to unconstraint large-vocabulary (more than 60,000 words) speech recognition. All the efficient search algorithms we discuss in this chapter and the next are considered as variants of dynamic programming or the Viterbi search algorithm.

In Section 12.1, cost (distance) is used as the measure of goodness for graph search algorithms. With Bayes' formulation, searching the minimum-cost path (word sequence) is equivalent to finding the path with maximum probability. For the sake of consistency, we use the inverse of Bayes' posterior probability as our objective function. Furthermore, logarithms are used on the inverse posterior probability to avoid multiplications. That is, the following new criterion is used to find the optimal word sequence $\hat{\mathbf{W}}$:

$$C(\mathbf{W}|\mathbf{X}) = \log\left[\frac{1}{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}\right] = -\log\left[P(\mathbf{W})P(\mathbf{X}|\mathbf{W})\right] \tag{12.6}$$

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} C(\mathbf{W}|\mathbf{X}) \tag{12.7}$$

For simplicity, we also define the following cost measures to mirror the likelihood for acoustic models and language models:

$$C(\mathbf{X}|\mathbf{W}) = -\log\left[P(\mathbf{X}|\mathbf{W})\right] \tag{12.8}$$

$$C(\mathbf{W}) = -\log\left[P(\mathbf{W})\right] \tag{12.9}$$

12.2.2. Combining Acoustic and Language Models

Although Bayes' equation [Eq. (12.5)] suggests that the acoustic model probability (conditional probability) and language model probability (prior probability) can be combined through simple multiplication, in practice some weighting is desirable. For example, when HMMs are used for acoustic models, the acoustic probability is usually underestimated, owing to the fallacy of the Markov and independence assumptions. Combining the language model probability with an underestimated acoustic model probability according to Eq. (12.5) would give the language model too little weight. Moreover, the two quantities have vastly different dynamic ranges particularly when continuous HMMs are used. One way to balance the two probability quantities is to add a *language model weight* LW to raise the language model probability $P(\mathbf{W})$ to that power $P(\mathbf{W})^{LW}$ [4, 25]. The language model weight LW is typically determined empirically to optimize the recognition performance on a development set. Since the acoustic model probabilities are underestimated, the language model weight LW is typically >1.

Language model probability has another function as a penalty for inserting a new word (or existing words). In particular, when a uniform language model (every word has an equal probability for any condition) is used, the language model probability here can be viewed as purely the penalty of inserting a new word. If this penalty is large, the decoder will prefer fewer longer words in general, and if this penalty is small, the decoder will prefer a greater number of shorter words instead. Since varying the language model weight to match the underestimated acoustic model probability will have some side effect of adjusting the penalty of inserting a new word, we sometimes use another independent *insertion penalty* to adjust the issue of longer or short words. Thus the language model contribution becomes:

$$P(\mathbf{W})^{LW} IP^{N(\mathbf{W})} \tag{12.10}$$

where IP is the insertion penalty (generally $0 < IP \leq 1.0$) and $N(\mathbf{W})$ is the number of words in sentence $\mathbf{W}$. According to Eq. (12.10), insertion penalty is generally a constant that is added to the negative-logarithm domain when extending the search to another new word. In Chapter 9, we described how to compute errors in a speech recognition system and introduced three types of error: substitutions, deletions and insertions. Insertion penalty is so named because it usually affects only insertions. Similar to language model weight, the insertion penalty is determined empirically to optimize the recognition performance on a development set.

12.2.3. Isolated Word Recognition

With isolated word recognition, word boundaries are known. If word HMMs are available, the acoustic model probability $P(\mathbf{X}|\mathbf{W})$ can be computed using the forward algorithm introduced in Chapter 8. The search becomes a simple pattern recognition problem, and the word

$\hat{W}$ with highest forward probability is then chosen as the recognized word. When subword models are used, word HMMs can be easily constructed by concatenating corresponding phoneme HMMs or other types of subword HMMs according to the procedure described in Chapter 9.

12.2.4. Continuous Speech Recognition

Search in continuous speech recognition is rather complicated, even for a small vocabulary, since the search algorithm has to consider the possibility of each word starting at any arbitrary time frame. Some of the earliest speech recognition systems took a two-stage approach towards continuous speech recognition, first hypothesizing the possible word boundaries and then using pattern matching techniques for recognizing the segmented patterns. However, due to significant cross-word co-articulation, there is no reliable segmentation algorithm for detecting word boundaries other than doing recognition itself.

Let's illustrate how you can extend the isolated-word search technique to continuous speech recognition by a simple example, as shown in Figure 12.10. This system contains only two words, w_1 and w_2. We assume the language model used here is an uniform unigram ($P(w_1) = P(w_2) = 1/2$).

It is important to represent the language structures in the same HMM framework. In Figure 12.10, we add one starting state S and one collector state C. The starting state has a null transition to the initial state of each word HMM with corresponding language model probability (1/2 in this case). The final state of each word HMM has a null transition to the collector state. The collector state then has a null transition back to the starting state in order to allow recursion. Similar to the case of embedding the phoneme (subword) HMMs into the word HMM for isolated speech recognition, we can embed the word HMMs for w_1 and w_2 into a new HMM corresponding to structure in Figure 12.10. Thus, the continuous speech search problem can be solved by the standard HMM formulations.

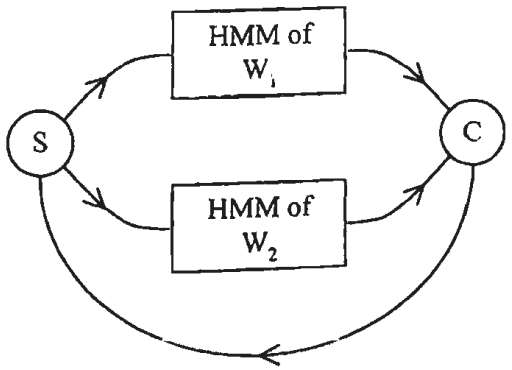


Figure 12.10 A simple example of continuous speech recognition task with two words w_1 and w_2. A uniform unigram language model is assumed for these words. State S is the starting state while state C is a collector state to save fully expanded links between every word pair.

The composite HMMs shown in Figure 12.10 can be viewed as a stochastic finite state network with transition probabilities and output distributions. The search algorithm is essentially producing a match between the acoustic observation $\mathbf{X}$ and a path[4] in the stochastic finite state network. Unlike isolated word recognition, continuous speech recognition needs to find the optimal word sequence $\hat{\mathbf{W}}$. The Viterbi algorithm is clearly a natural choice for this task since the optimal state sequence $\hat{\mathbf{S}}$ corresponds to the optimal word sequence $\hat{\mathbf{W}}$. Figure 12.11 shows the HMM Viterbi trellis computation for the two-word continuous speech recognition example in Figure 12.10. There is a cell for each state in the stochastic finite state network and each time frame t in the trellis. Each cell $C_{s,t}$ in the trellis can be connected to a cell corresponding to time t or $t+1$ and to states in the stochastic finite state network that can be reached from s. To make a word transition, there is a null transition to connect the final state of each word HMM to the initial state of the next word HMM that can be followed. The trellis computation is done *time-synchronously* from left to right, i.e., each cell for time t is completely computed before proceeding to time $t+1$.

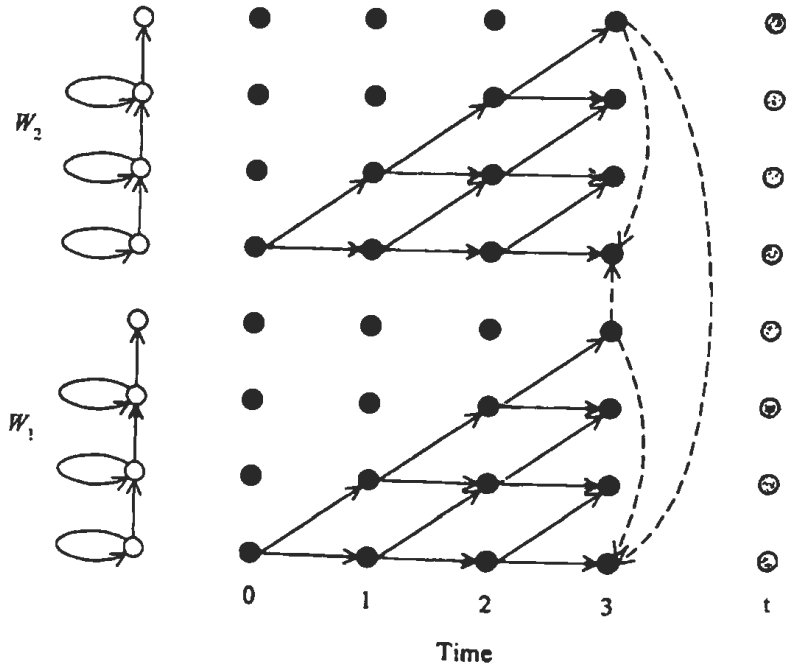


Figure 12.11 HMM trellis for continuous speech recognition example in Figure 12.10. When the final state of the word HMM is reached, a null arc (indicated by a dashed line) is linked from it to the initial state of the following word.

[4] A path here means a sequence of states and transitions.

12.3. LANGUAGE MODEL STATES

The state-space is a good indicator of search complexity. Since the HMM representation for each word in the lexicon is fixed, the state-space is determined by the language models. According to Chapter 11, every language model (grammar) is associated with a state machine (automata). Such a state machine is expanded to form the state-space for the recognizer. The states in such a state machine are referred to as *language models states*. For simplicity, we will use the concepts of state-space and language model states interchangeably. The expansion of language model states to HMM states will be done implicitly. The language model states for isolated word recognition are trivial. They are just the union of the HMM states of each word. In this section we look at the language model states for various grammars for continuous speech recognition.

12.3.1. Search Space with FSM and CFG

As described in Chapter 8, the complexity for the Viterbi algorithm is $O(N^2T)$, where N is the total number of states in the composite HMM and T is the length of input observation. A full time-synchronous Viterbi search is quite efficient for moderate tasks (vocabulary $\leq$ 500). We have already demonstrated in Figure 12.11 how to search for a two-word continuous speech recognition task with a uniform language model. The uniform language model, which allows all words in the vocabulary to follow every word with the same probability, is suitable for connected-digit task. In fact, most small vocabulary tasks in speech recognition applications usually use a finite state grammar (FSG).

Figure 12.12 shows a simple example of an FSM. Similar to the process described in Sections 12.2.3 and 12.2.4, each of the word arcs in an FSG can be expanded as a network of phoneme (subword) HMMs. The word HMMs are connected with null transitions with the grammar state. A large finite state HMM network that encodes all the legal sentences can be constructed based on the expansion procedure. The decoding process is achieved by performing a time-synchronous Viterbi search on this composite finite state HMM.

In practice, FSGs are sufficient for simple tasks. However, when an FSG is made to satisfy the constraints of sharing of different sub-grammars for compactness and support for dynamic modifications, the resulting non-deterministic FSG is very similar to context-free grammar (CFG) in terms of implementation. The CFG grammar consists of a set of productions or rules, which expand nonterminals into a sequence of terminals and nonterminals. Nonterminals in the grammar tend to refer to high-level task-specific concepts such as dates, names, and commands. The terminals are words in the vocabulary. A grammar also has a non-terminal designated as its start state.

Although efficient parsing algorithms, like chart parsing (described in Chapter 11), are available for CFG, they are not suitable for speech recognition, which requires left-to-right processing. A context-free grammar can be formulated with a recursive transition network (RTN). RTNs are more powerful and complicated than the finite state machines described in

Chapter 11 because they allow arc labels to refer to other networks as well as words. We use Figure 12.13 to illustrate how to embed HMMs into a recursive transition network.

Figure 12.13 is an RTN representation of the following CFG:

```
S→ NP VP
NP→ sam | sam davis
VP → VERB tom
VERB → likes | hates
```

There are three types of arcs in an RTN, as shown in Figure 12.13: CAT(x), PUSH (x), and POP(x). The CAT(x) arc indicates that x is a terminal node (which is equivalent to a word arc). Therefore, all the CAT(x) arcs can be expanded by the HMM network for x. The word HMM can again be a composite HMM built from phoneme (or subword) HMMs. Similar to the finite state grammar case in Figure 12.12, each grammar state acts as a state with incoming and outgoing null transitions to connect word HMMs in the CFG.

During decoding, the search pursues several paths through the CFG at the same time. Associated with each of the paths is a grammar state that describes completely how the path can be extended further. When the decoder hypothesizes the end of the current word of a path, it asks the CFG module to extend the path further by one word. There may be several alternative successor words for the given path. The decoder considers all the successor word possibilities. This may cause the path to be extended to generate several more paths to be considered, each with its own grammar state.

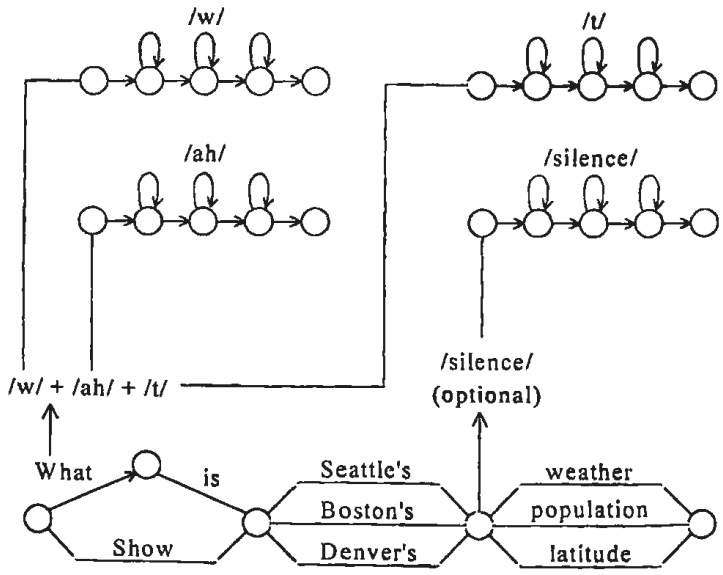


Figure 12.12 An illustration of how to compile a speech recognition task with finite state grammar into a composite HMM.

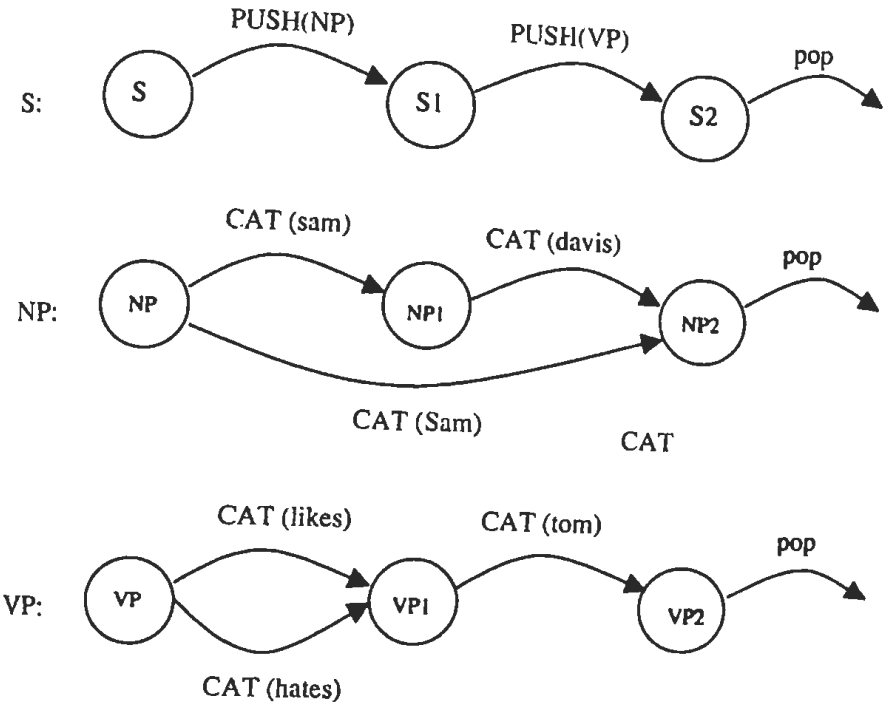


Figure 12.13 A simple RTN example with three types of arcs: CAT(x), PUSH (x), POP.

Readers should note that the same word might be under consideration by the decoder in the context of different paths and grammar states at the same time. For example, there are two word arcs CAT (Sam) in Figure 12.13. Their HMM states should be considered as distinct states in the trellis because they are in completely different grammar states. Two different states in the trellis also means that different paths going into these two states cannot be merged. Since these two partial paths will lead to different successive paths, the search decision needs to be postponed until the end of search. Therefore, when embedding HMMs into word arcs in the grammar network, the HMM state will be assigned a new state identity, although the HMM parameters (transition probabilities and output distributions) can still be shared across different grammar arcs.

Each path consists of a stack of production rules. Each element of the stack also contains the position within the production rule of the symbol that is currently being explored. The search graph (trellis) started from the initial state of CFG (state S). When the path needs to be extended, we look at the next arc (symbol in CFG) in the production. When the search enters a CAT(x) arc (terminal), the path gets extended with the terminal, and the HMM trellis computation is performed on the CAT(x) arc to match the model x against the acoustic data. When the final state of the HMM for x is reached, the search moves on via the null

transition to the destination of the CAT(x) arc. When the search enters a PUSH(x) arc, it indicates a nonterminal symbol x is encountered. In effect, the search is about to enter a subnetwork of x; the destination of the PUSH(x) arc is stored in a last-in first-out (LIFO) stack. When the search reaches a POP arc that signals the end of the current network, the control should jump back to the calling network. In other words, the search returns to the state extracted from the top of the LIFO stack. Finally, when we reach the end of the production rule at the very bottom of the stack, we have reached an accepting state in which we have seen a complete grammatical sentence. For our decoding purpose, that is the state we want to pick as the best score at the end of time frame T to get the search result.

The problem of connected word recognition by finite state or context-free grammars is that the number of states increases enormously when it is applied to more complex grammars. Moreover it remains a challenge to generate such FSGs or CFGs from a large corpus, either manually or automatically. As mentioned in Chapter 11, it is questionable whether FSG or CFG is adequate to describe natural languages or unconstrained spontaneous languages. Instead, n-gram language models are often used for natural languages or unconstrained spontaneous languages. In the next section we investigate how to integrate various n-grams into continuous speech recognition.

12.3.2. Search Space with the Unigram

The simplest n-gram is the unigram that is memory-less and depends only on the current word.

$$P(\mathbf{W}) = \prod_{i=1}^{n} P(w_i) \tag{12.11}$$

Figure 12.14 shows such a unigram grammar network. The final state of each word HMM is connected to the collector state by a null transition, with probability 1.0. The collector state is then connected to the starting state by another null transition, with transition probability equal to 1.0. For word expansion, the starting state is connected to the initial state of each word HMM by a null transition, with transition probability equal to the corresponding unigram probability. Using the collector state and starting state for word expansion allows efficient expansion because it first merges all the word-ending paths[5] (only the best one survives) before expansion. It can cut the total cross-word expansion from N^2 to N.

[5] In graph search, a partial path still under consideration is also referred to as a theory, although we will use paths instead of theories in this book.

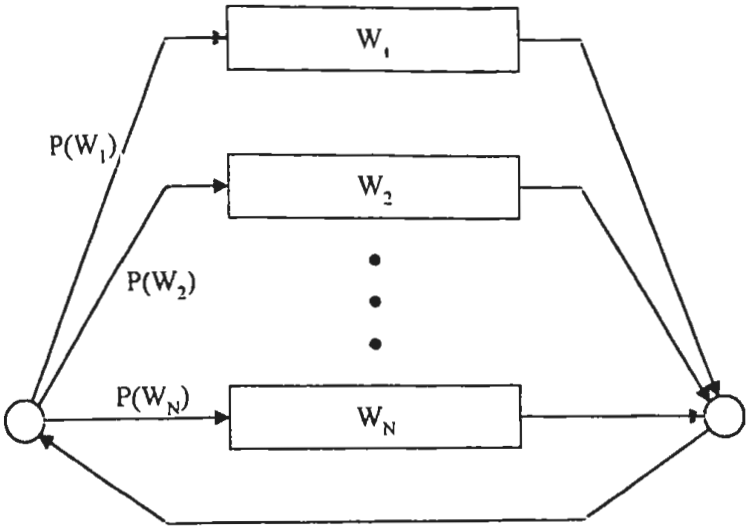


Figure 12.14 A unigram grammar network where the unigram probability is attached as the transition probability from starting state S to the first state of each word HMM.

12.3.3. Search Space with Bigrams

When the bigram is used, the probability of a word depends only on the immediately preceding word. Thus, the language model score is:

$$P(\mathbf{W}) = P(w_1 \mid <s>)\prod_{i=2}^{n} P(w_i \mid w_{i-1}) \tag{12.12}$$

where <s> represents the symbol of starting of a sentence.

Figure 12.15 shows a grammar network using a bigram language model. Because of the bigram constraint, the merge-and-expand framework for unigram search no longer applies here. Instead, the bigram search needs to perform expand-and-merge. Thus, bigram expansion is more expensive than unigram expansion. For a vocabulary size N, the bigram would need N^2 word-to-word transitions in comparison to N for the unigram. Each word transition has a transition probability equal to the corresponding bigram probability. Fortunately, the total number of states for bigram search is still proportional to the vocabulary size N.

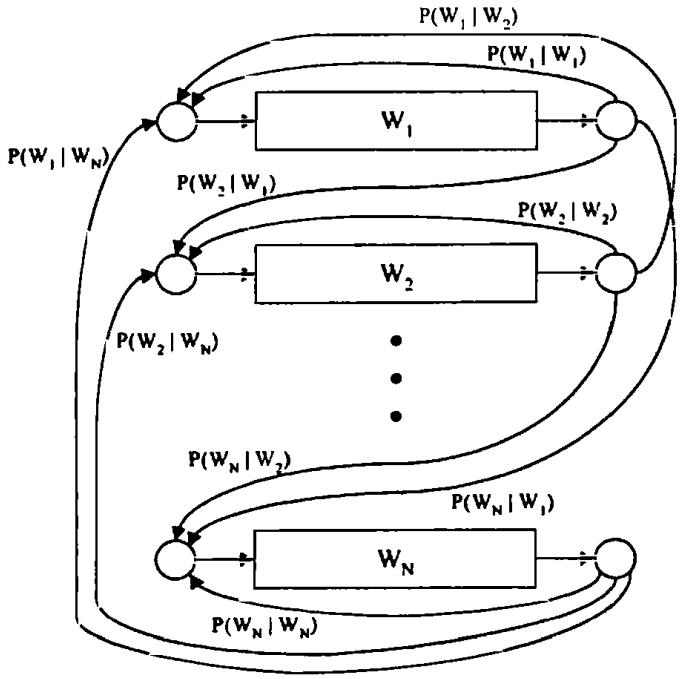


Figure 12.15 A bigram grammar network where the bigram probability $P(w_j \mid w_i)$ is attached as the transition probability from word w_i to w_j [19].

Because the search space for bigram is kept manageable, bigram search can be implemented very efficiently. Bigram search is a good compromise between efficient search and effective language models. Therefore, bigram search is arguably the most widely used search technique for unconstrained large-vocabulary continuous speech recognition. Particularly for the multiple-pass search techniques described in Chapter 13, a bigram search is often used in the first pass search.

12.3.3.1. Backoff Paths

When the vocabulary size N is large, the total bigram expansion N^2 can become computationally prohibitive. As described in Chapter 11, only a limited number of bigrams are observable in any practical corpora for a large vocabulary size. Suppose the probabilities for unseen bigrams are obtained through Katz's backoff mechanism. That is, for unseen bigram $P(w_j \mid w_i)$,

$$P(w_j \mid w_i) = \alpha(w_i) P(w_j) \tag{12.13}$$

where $\alpha(w_i)$ is the backoff weight for word w_i.

Using the backoff mechanism for unseen bigrams, the bigram expansion can be significantly reduced [12]. Figure 12.16 shows the new word expansion scheme. Instead of full bigram expansion, only observed bigrams are connected by direct word transitions with correspondent bigram probabilities. For backoff bigrams, the last state of word w_i is first connected to a central backoff node with transition probability equal to backoff weight $\alpha(w_i)$. The backoff node is then connected to the beginning of each word w_j with transition probability equal to its corresponding unigram probability $P(w_j)$. Readers should note that there are now two paths from w_i to w_j for an observed bigram $P(w_j \mid w_i)$. One is the direct link representing the observable bigram $P(w_j \mid w_i)$, and the other is the two-link backoff path representing $\alpha(w_i)P(w_j)$. For a word pair whose bigram exists, the two-link backoff path is likely to be ignored since the backoff unigram probability is almost always smaller than the observed bigram $P(w_j \mid w_i)$. Suppose there are only N_b different observable bigrams, this scheme requires $N_b + 2N$ instead of N^2 word transitions. Since under normal circumstance $N_b \ll N^2$, this backoff scheme significantly reduces the cost of word expansion.

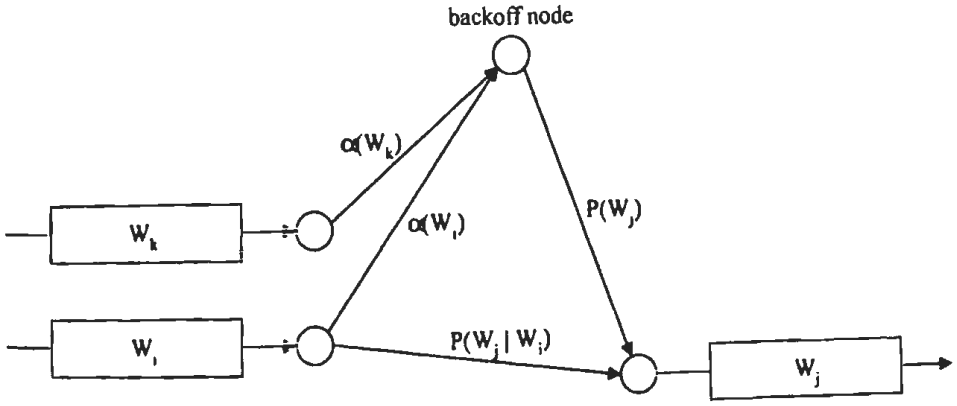


Figure 12.16 Reducing bigram expansion in a search by using the backoff node. In addition to normal bigram expansion arcs for all observed bigrams, the last state of word w_i is first connected to a central backoff node with transition probability equal to backoff weight $\alpha(w_i)$. The backoff node is then connected to the beginning of each word w_j with its corresponding unigram probability $P(w_j)$ [12].

12.3.4. Search Space with Trigrams

For a trigram language model, the language model score is:

$$P(\mathrm{W}) = P(w_1 \mid <\mathrm{s}>)P(w_2 \mid <\mathrm{s}>, w_1)\prod_{i=3}^{n} P(w_i \mid w_{i-2}, w_{i-1}) \tag{12.14}$$

The search space is considerably more complex, as shown in Figure 12.17. Since the equivalence grammar class is the previous two words w_i and w_j, the total number of grammar states is N^2. From each of these grammar states, there is a transition to the next word [19].

Obviously, it is very expensive to implement large-vocabulary trigram search given the complexity of the search space. It becomes necessary to dynamically generate the trigram search graph (trellis) via a graph search algorithm. The other alternative is to perform a multiple-pass search strategy, in which the first-pass search uses less detailed language models, like bigrams, to generate an n-best list or word lattice, and then a second-pass detailed search can use trigrams on a much smaller search space. Multiple-pass search strategy is discussed in Chapter 13.

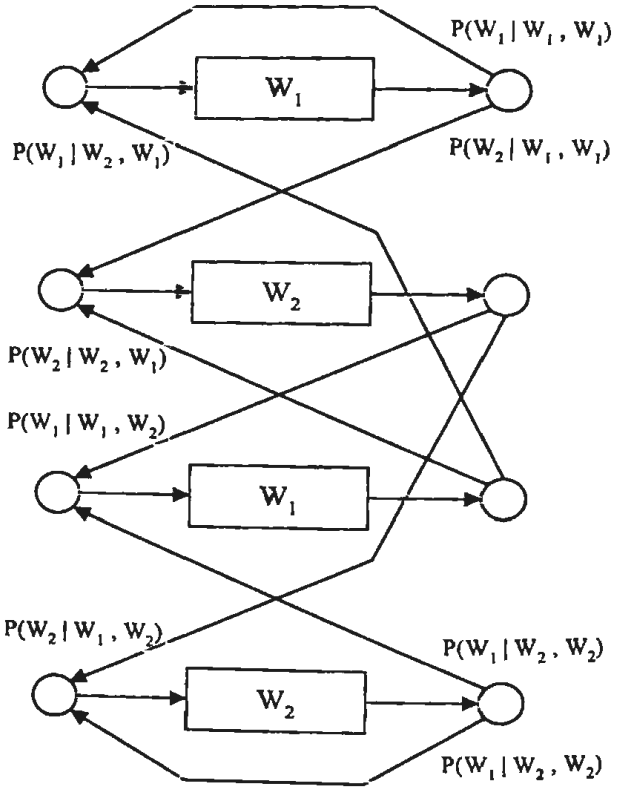


Figure 12.17 A trigram grammar network where the trigram probability $P(w_k \mid w_i, w_j)$ is attached to transition from grammar state word w_i, w_j to the next word w_k. Illustrated here is a two-word vocabulary, so there are four grammar states in the trigram network [19].

12.3.5. How to Handle Silences Between Words

In continuous speech recognition, there are unavoidable pauses (silences) between words or sentences. The pause is often referred to as silence or a non-speech event in continuous speech recognition. Acoustically, the pause is modeled by a silence model[6] that models background acoustic phenomena. The silence model is usually modeled with a topology flexible enough to accommodate a wide range of lengths, since the duration of a pause is arbitrary.

It can be argued that silences are actually linguistically distinguishable events, which contribute to prosodic and meaning representation. For example, people are likely to pause more often in phrasal boundaries. However, these patterns are so far not well understood for unconstrained natural speech (particularly for spontaneous speech). Therefore, the design of almost all automatic speech recognition systems today allows silences occurring just about anywhere between two lexical tokens or between sentences. It is relatively safe to assume that people pause a little bit between sentences to catch breath, so the silences between sentences are assumed mandatory while silences between words are optional. In most systems, silence is often modeled as a special lexicon entry with special language model probability. This special language model probability is also referred to as silence insertion penalty that is set to adjust the likelihood of inserting such an optional silence between words.

It is relatively straightforward to handle the optional silence between words. We need only to replace all the grammar states connecting words with a small network like the one shown in Figure 12.18. This arrangement is similar to that of the optional silence in training continuous speech, described in Chapter 9. The small network contains two parallel paths. One is the original null transition acting as the direct transition from one word to another, while the other path will need to go through a silence model with the silence insertion penalty attached in the transition probability before going to the next word.

One thing to clarify in the implementation of Figure 12.18 is that this silence expansion needs to be done for every grammar state connecting words. In the unigram grammar network of Figure 12.14, since there is only one collector node to connect words, the silence expansion is required only for this collector node. On the other hand, in the bigram grammar network of Figure 12.15, there is a collector node for every word before expanding to the next word. In this case, the silence expansion is required for every collector node. For a vocabulary size $|V|$, this means there are $|V|$ numbers of silence networks in the grammar search network. This requirement lies in the fact that in bigram search we cannot merge paths before expanding into the next word. Optional silence can then be regarded as part of the search effort for the previous word, so the word expansion needs to be done after finishing the optional silence. Therefore, we treat each word as having two possible pronunciations, one with the silence at the end and one without. This viewpoint integrates silence in the word pronunciation network like the example shown in Figure 12.19.

[6] Some researchers extend the context-dependent modeling to silence models. In that case, there are several silence models based on surrounding contexts.

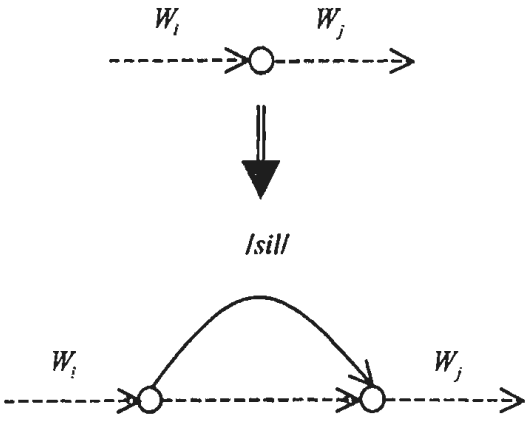


Figure 12.18 Incorporating optional silence (a non-speech event) in the grammar search network where the grammar state connecting different words is laced by two parallel paths. One is the original null transition directly from one word to the other, while the other first goes through the silence word to accommodate the optional silence.

For efficiency reasons, a single silence is sometimes used for large-vocabulary continuous speech recognition using higher order n-gram language model. Theoretically, this could be a source of pruning errors.[7] However, the error could turn out to be so small as to be negligible because there are, in general, very few pauses between word for continuous speech. On the other hand, the overhead of using multiple silences should be very minimal because it is less likely to visit those silence models at the end of words due to pruning.

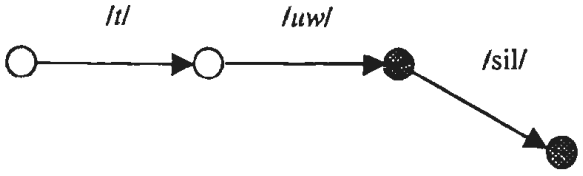


Figure 12.19 An example of treating silence as of the pronunciation network of word TWO. The shaded nodes represent possible word-ending nodes: one without silence and the other one with silence.

12.4. TIME-SYNCHRONOUS VITERBI BEAM SEARCH

When HMMs are used for acoustic models, the acoustic model score (likelihood) used in search is by definition the forward probability. That is, all possible state sequences must be considered. Thus,

[7] Speech recognition errors due to sub-optimal search or heuristic pruning are referred to as *pruning errors*, which will be described in detail in Chapter 13.

$$P(\mathbf{X}|\mathbf{W}) = \sum_{\text{all possible } s_0^T} P(\mathbf{X}, s_0^T | \mathbf{W}) \tag{12.15}$$

where the summation is to be taken over all possible state sequences S with the word sequence W under consideration. However, under the trellis framework (as in Figure 12.11), more bookkeeping must be performed since we cannot add scores with different word sequence history. Since the goal of decoding is to uncover the best word sequence, we could approximate the summation with the maximum to find the best state sequence instead. The Bayes' decision rule, Eq. (12.5), becomes

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X}|\mathbf{W}) \cong \arg\max_{\mathbf{W}} \left\{ P(\mathbf{W}) \max_{s_0^T} P(\mathbf{X}, s_0^T | \mathbf{W}) \right\} \tag{12.16}$$

Equation (12.16) is often referred to as the *Viterbi approximation*. It can be literally translated to "the *most likely word sequence* is approximated by the *most likely state sequence*." Viterbi search is then sub-optimal. Although the search results by using forward probability and Viterbi probability could, in principle, be different, in practice this is rarely the case. We use this approximation for the rest of this chapter.

The Viterbi search has already been discussed as a solution to one of the three fundamental HMM problems in Chapter 8. It can be executed very efficiently via the same trellis framework. To briefly reiterate, the Viterbi search is a time-synchronous search algorithm that completely processes time t before going on to time $t+1$. For time t, each state is updated by the best score (instead of the sum of all incoming paths) from all states in at time $t-1$. This is why it is often called *time-synchronous Viterbi search*. When one update occurs, it also records the backtracking pointer to remember the most probable incoming state. At the end of search, the most probable state sequence can be recovered by tracing back these backtracking pointers. The Viterbi algorithm provides an optimal solution for handling nonlinear time warping between hidden Markov models and acoustic observation, word boundary detection and word identification in continuous speech recognition. This unified Viterbi search algorithm serves as the basis for all search algorithms as described in the rest of the chapter.

It is necessary to clarify the backtracking pointer for time-synchronous Viterbi search for continuous word recognition. We are generally not interested in the optimal state sequence for speech recognition.[8] Instead, we are only interested in the optimal word sequence indicated by Eq. (12.16). Therefore, we use the backtrack pointer just to remember the word history for the current path, so the optimal word sequence can be recovered at the end of search. To be more specific, when we reach the final state of a word, we create a history node containing the word identity and current time index and append this history node to the existing backtrack pointer. This backtrack pointer is then passed onto the successor node if it

[8] While we are not interested in optimal state sequences for ASR, they are very useful in deriving phonetic segmentation, which could provide important information for developing ASR systems.

is the optimal path leading to the successor node for both intra-word and inter-word transition. The side benefit of keeping this backtrack pointer is that we no longer need to keep the entire trellis during the search. Instead, we only need space to keep two successive time slices (columns) in the trellis computation (the previous time slice and the current time slice) because all the backtracking information is now kept in the backtrack pointer. This simplification is a significant benefit in the implementation of a time-synchronous Viterbi search.

Time-synchronous Viterbi search can be considered as a *breadth-first search* with dynamic programming. Instead of performing a tree search algorithm, the dynamic programming principle helps create a search graph where multiple paths leading to the same search state are merged by keeping the best path (with minimum cost). The Viterbi trellis is a representation of the search graph. Therefore, all the efficient techniques for graph search algorithms can be applied to time-synchronous Viterbi search. Although so far we have described the trellis in an explicit fashion—the whole search space needs to be explored before the optimal path can be found—it is not necessary to do so. When the search space contains an enormous number of states, it becomes impractical to pre-compile the composite HMM entirely and store it in the memory. It is preferable to dynamically build and allocate portions of the search space sufficient to search the promising paths. By using the graph search algorithm described in Section 12.1.1, only part of the entire Viterbi trellis is generated explicitly. By constructing the search space dynamically, the computation cost of the search is proportional only to the number of active hypotheses, independent of the overall size of the potential search space. Therefore, dynamically generated trellises are key to heuristic Viterbi search for efficient large-vocabulary continuous speech recognition, as described in Chapter 13.

12.4.1. The Use of Beam

Based on Chapter 8, the search space for Viterbi search is $O(NT)$ and the complexity is $O(N^2T)$, where N is the total number of HMM states and T is the length of the utterance. For large-vocabulary tasks these numbers are astronomically large even with the help of dynamic programming. In order to avoid examining the overwhelming number of possible cells in the HMM trellis, a heuristic search is clearly needed. Different heuristics generate or explore portions of the trellis in different ways.

A simple way to prune the search space for breadth-first search is the beam search described in Section 12.1.3.2. Instead of retaining all candidates (cells) at every time frame, a threshold T is used to keep only a subset of promising candidates. The state at time t with the lowest cost D_{min} is first identified. Then each state at time t with cost $> D_{min} + T$ is discarded from further consideration before moving on to the next time frame $t+1$. The use of the beam alleviates the need to process all the cells. In practice, it can lead to substantial savings in computation with little or no loss of accuracy.

Although beam search is a simple idea, the combination of time-synchronous Viterbi and beam search algorithms produces the most powerful search strategy for large-vocabulary speech recognition. Comparing paths with equal length under a time-

synchronous search framework makes beam search possible. That is, for two different word sequences $\mathbf{W}_1$ and $\mathbf{W}_2$, the posterior probabilities $P(\mathbf{W}_1 \mid \mathbf{x}_0^t)$ and $P(\mathbf{W}_2 \mid \mathbf{x}_0^t)$ are always compared based on the same partial acoustic observation $\mathbf{x}_0^t$. This makes the comparison straightforward because the denominator $P(\mathbf{x}_0^t)$ in Eq. (12.5) is the same for both terms and can be ignored. Since the score comparison for each time frame is fair, the only assumption of beam search is that an optimal path should have a good enough partial-path score for each time frame to survive under beam pruning.

The time-synchronous framework is one of the aspects of Viterbi beam search that is critical to its success. Unlike the time-synchronous framework, time-asynchronous search algorithms such as stack decoding require the normalization of likelihood scores over feature streams of different time lengths. This, as we will see in Section 12.5, is the Achilles' heel of that approach.

The straightforward time-synchronous Viterbi beam search is ineffective in dealing with the gigantic search space of high perplexity tasks. However, with a better understanding of the linguistic search space and the advent of techniques for obtaining n-best lists from time-synchronous Viterbi search, described in Chapter 13, time-synchronous Viterbi beam search has turned out to be surprisingly successful in handling tasks of all sizes and all different types of grammars, including FSG, CFG, and n-gram [2, 14, 18, 28, 34, 38, 44]. Therefore, it has become the predominant search strategy for continuous speech recognition.

12.4.2. Viterbi Beam Search

To explain the time-synchronous Viterbi beam search in a formal way [31], we first define some quantities:

$D(t;s_t;w) \equiv$ total cost of the best path up to time t that ends in state s_t of grammar word state w.

$h(t;s_t;w) \equiv$ backtrack pointer for the best path up to time t that ends in state s_t of grammar word state w.

Readers should be aware that w in the two quantities above represents a grammar word state in the search space. It is different from just the word identity since the same word could occur in many different language model states, as in the trigram search space shown in Figure 12.17.

There are two types of dynamic programming (DP) transition rules [30], namely intra-word and inter-word transition. The intra-word transition is just like the Viterbi rule for HMMs and can be expressed as follows:

$$D(t;s_t;w) = \min_{s_{t-1}} \{d(\mathbf{x}_t, s_t \mid s_{t-1};w) + D(t-1;s_{t-1};w)\} \tag{12.17}$$

$$h(t;s_t;w) = h(t-1, b_{\min}(t;s_t;w);w) \tag{12.18}$$

where $d(\mathbf{x}_t, s_t \mid s_{t-1}; w)$ is the cost associated with taking the transition from state s_{t-1} to state s_t while generating output observation $\mathbf{x}_t$, and $b_{min}(t; s_t; w)$ is the optimal predecessor state of cell $D(t; s_t; w)$. To be specific, they can be expressed as follows:

$$d(\mathbf{x}_t, s_t \mid s_{t-1}; w) = -\log P(s_t \mid s_{t-1}; w) - \log P(\mathbf{x}_t \mid s_t; w) \tag{12.19}$$

$$b_{min}(t; s_t; w) = \arg\min_{s_{t-1}} \{d(\mathbf{x}_t, s_t \mid s_{t-1}; w) + D(t-1; s_{t-1}; w)\} \tag{12.20}$$

The inter-word transition is basically a null transition without consuming any observation. However, it needs to deal with creating a new history node for the backtracking pointer. Let's define $F(w)$ as the final state of word HMM w and $I(w)$ as the initial state of word HMM w. Moreover, state η is denoted as the pseudo initial state. The inter-word transition can then be expressed as follows:

$$D(t; \eta; w) = \min_v \{\log P(w \mid v) + D(t; F(v); v)\} \tag{12.21}$$

$$h(t; \eta; w) = \langle v_{min}, t \rangle :: h(t, F(v_{min}); v_{min}) \tag{12.22}$$

where $v_{min} = \arg\min_v \{\log P(w \mid v) + D(t; F(v); v)\}$ and :: is a link appending operator.

The time-synchronous Viterbi beam search algorithm assumes that all the intra-word transitions are evaluated before inter-word null transitions take place. The same time index is used intentionally for inter-word transition since the null language model state transition does not consume an observation vector. Since the initial state $I(w)$ for word HMM w could have a self-transition, the cell $D(t; I(w); w)$ might already have an active path. Therefore, we need to perform the following check to advance the inter-word transitions.

$$\begin{aligned} &\text{if } D(t; \eta; w) < D(t; I(w); w) \\ &\quad D(t; I(w); w) = D(t; \eta; w) \text{ and } h(t; I(w); w) = h(t; \eta; w) \end{aligned} \tag{12.23}$$

The time-synchronous Viterbi beam search can be summarized as in Algorithm 12.6. For large-vocabulary speech recognition, the experimental results show that only a small percentage of the entire search space (the beam) needs to be kept for each time interval t without increasing error rates. Empirically, the beam size has typically been found to be between 5% and 10% of the entire search space. In Chapter 13 we describe strategies of using different level of beams for more effectively pruning.

12.5. STACK DECODING (A* SEARCH)

If some reliable heuristics are available to guide the decoding, the search can be done in a depth-first fashion around the best path early on, instead of wasting efforts on unpromising paths via the time-synchronous beam search. Stack decoding represents the best attempt to

ALGORITHM 12.6: TIME-SYNCHRONOUS VITERBI BEAM SEARCH

Step 1: Initialization: For all the grammar word states w which can start a sentence,

$$D(0; I(w); w) = 0$$

$$h(0; I(w); w) = null$$

Step 2: Induction: For time $t = 1$ to T do

For all active states do

Intra-word transitions according to Eq. (12.17) and (12.18)

$$D(t; s_t; w) = \min_{s_{t-1}} \left\{ d(\mathbf{x}_t, s_t \mid s_{t-1}; w) + D(t-1; s_{t-1}; w) \right\}$$

$$h(t; s_t; w) = h(t-1, b_{\min}(t; s_t; w); w)$$

For all active word-final states do

Inter-word transitions according to Eq. (12.21), (12.22) and (12.23)

$$D(t; \eta; w) = \min_{v} \left\{ \log P(w \mid v) + D(t; F(v); v) \right\}$$

$$h(t; \eta; w) = \langle v_{\min}, t \rangle :: h(t, F(v_{\min}); v_{\min})$$

if $D(t; \eta; w) < D(t; I(w); w)$

$$D(t; I(w); w) = D(t; \eta; w) \text{ and } h(t; I(w); w) = h(t; \eta; w)$$

Pruning: Find the cost for the best path and decide the beam threshold

Prune unpromising hypotheses

Step 3: Termination: Pick the best path among all the possible final states of grammar at time T. Obtain the optimal word sequence according to the backtracking pointer $h(t; \eta; w)$

use A* search instead of time-synchronous beam search for continuous speech recognition. Unfortunately, as we will discover in this section, such a heuristic function $h(\bullet)$ (defined in Section 12.1.3) is very difficult to attain in continuous speech recognition, so search algorithms based on A* search are in general less efficient than time-synchronous beam search.

Stack decoding is a variant of the heuristic A* search based on the forward algorithm, where the evaluation function is based on the forward probability. It is a tree search algorithm, which takes a slightly different viewpoint than the time-synchronous Viterbi search. Time-synchronous beam search is basically a breadth-first search, so it is crucial to control the number of all possible language model states as described in Section 12.3. In a typical large-vocabulary Viterbi search with n-gram language models, this number is determined by the equivalent classes of language model histories. On the other hand, stack decoding as a tree search algorithm treats the search as a task for finding a path in a tree whose branches correspond to words in the vocabulary V, non-terminal nodes correspond to incomplete sentences, and terminal nodes correspond to complete sentences. The search tree has a constant branching factor of $|V|$, if we allow every word to be followed by every word. Figure 12.20 illustrates such a search tree for a vocabulary with three words [19].

An important advantage of stack decoding is its consistency with the forward-backward training algorithm. Viterbi search is a graph search, and paths cannot be easily summed because they may have different word histories. In general, the Viterbi search finds the optimal state sequence instead of optimal word sequence. Therefore, Viterbi approximation is necessary to make the Viterbi search feasible, as described in Section 12.4. Stack decoding is a tree search, so each node has a unique history, and the forward algorithm can be used within word model evaluation. Moreover, all possible beginning and ending times (shaded areas in Figure 12.21) are considered [24]. With stack decoding, it is possible to use an objective function that searches for the optimal word string, rather than the optimal state sequence. Furthermore, it is in principle natural for stack decoding to accommodate long-range language models if the heuristics can guide the search to avoid exploring the overwhelmingly large unpromising grammar states.

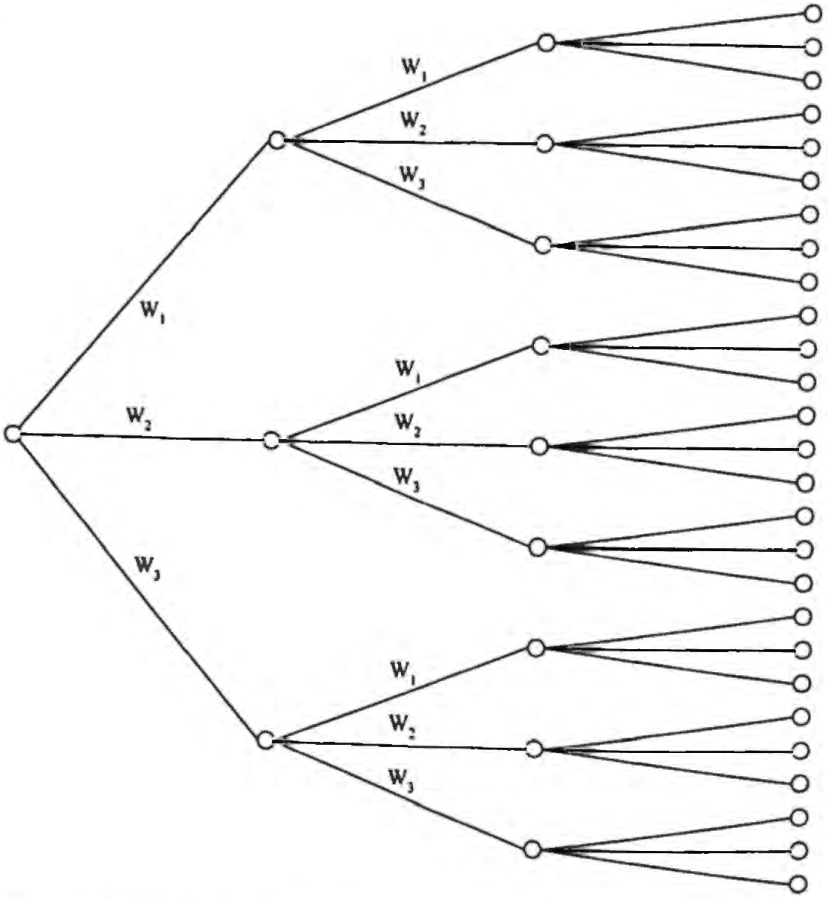


Figure 12.20 A stack decoding search tree for a vocabulary size of three [19].

By formulating stack decoding in a tree search framework, the graph search algorithms described in Section 12.1 can be directly applied to stack decoding. Obviously, blind-search methods, like depth-first and breadth-first search, that do not take advantage of the goodness measurement of how close we are getting to the goal, are usually computationally infeasible in practical speech recognition systems. A* search is clearly attractive for speech recognition, given the hope of a sufficient heuristic function to guide the tree search in a favorable direction without exploring too many unpromising branches and nodes. In contrast to the Viterbi search, it is not time-synchronous and extends paths of different lengths.

The search begins by adding all possible one-word hypotheses to the *OPEN* list. Then the best path is removed from the *OPEN* list, and all paths from it are extended, evaluated, and placed back in the *OPEN* list. This search continues until a complete path that is guaranteed to be better than all paths in the *OPEN* list has been found.

Unlike Viterbi search, where the acoustic probabilities being compared are always based on the same partial input, it is necessary to compare the goodness of partial paths of different lengths to direct the A* tree search. Moreover, since stack decoding is done asynchronously, we need an effective mechanism to determine when to end a phone/word evaluation and move on to the next phone/word. Therefore, the heart and soul of the stack decoding are clearly in

1. Finding an effective and efficient heuristic function for estimating the future remaining input feature stream and
2. Determining when to extend the search to the next word/phone.

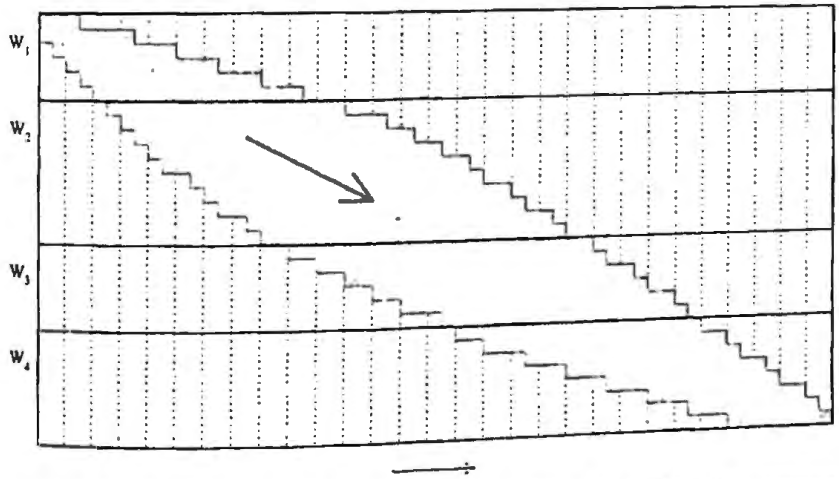


Figure 12.21 The forward trellis space for stack decoding. Each grid point corresponds to a trellis cell in the forward computation. The shaded area represents the values contributing to the computation of the forward score for the optimal word sequence $w_1, w_2, w_3, \ldots$ [24].

In the following section we describe these two critical components. Readers will note that the solutions to these two issues are virtually the same—using a normalization scheme to compare paths of different lengths.

12.5.1. Admissible Heuristics for Remaining Path

The key issue in heuristic search is the selection of an evaluation function. As described in Section 12.1.3, the heuristic function of the path H_N going through node N includes the cost up to the node and the estimate of the cost to the target node from node N. Suppose path H_N is going through node N at time t; then the evaluation for path H_N can be expressed as follows:

$$f(H_N^t) = g(H_N^t) + h(H_N^{t,T}) \tag{12.24}$$

where $g(H_N^t)$ is the evaluation function for the partial path of H_N up to time t, and $h(H_N^{t,T})$ is the heuristic function of the remaining path from $t+1$ to T for path H_N. The challenge for stack decoders is to devise an admissible function for $h(\bullet)$.

According to Section 12.1.3.1, an admissible heuristic function is one that always underestimates the true cost of the remaining path from $t+1$ to T for path H_N. A trivially admissible function is the zero function. In this case, it results in a very large OPEN list. In addition, since the longer paths tend to have higher cost because of the gradually accumulated cost, the search is likely to be conducted in a breadth-first fashion, which functions very much like a plain Viterbi search. The evaluation function $g(\bullet)$ can be obtained easily by using the HMM forward score as the true cost up to current time t. However, how can we find an admissible heuristic function $h(\bullet)$? We present the basic concept here [19, 35].

The goal of $h(\bullet)$ is to find the expected cost for the remaining path. If we can obtain the expected cost per frame ψ for the remaining path, the total expected cost, $(T-t)*\psi$, is simply the product of ψ and the length of the remaining path. One way to find such expected cost per frame is to gather statistics empirically from training data.

1. After the final training iteration, perform Viterbi forced alignment[9] with each training utterance to get an optimal time alignment for each word.
2. Randomly select an interval to cover the number of words ranging from two to ten. Denote this interval as $[i \ldots j]$.
3. Compute the average acoustic cost per frame within this selected interval according to the following formula and save the value in a set Λ:

[9] Viterbi forced alignment means that the Viterbi is performed on the HMM model constructed from the known word transcription. The term "forced" is used because the Viterbi alignment is forced to be performed on the correct model. Viterbi forced alignment is a very useful tool in spoken language processing as it can provide the optimal state-time alignment with the utterances. This detailed alignment can then be used for different purposes, including discriminant training, concatenated speech synthesis, etc.

$$\frac{-1}{j-i}\log P(\mathbf{x}_i^j \mid \mathbf{w}_{i\ldots j}) \tag{12.25}$$

where $\mathbf{w}_{i\ldots j}$ is the word string corresponding to interval $[i\ldots j]$.

4. Repeat Steps 2 and 3 for the entire training set.

5. Define $\psi_{\min}$ and ψ_{avg} as the minimum and average value found in set Λ.

Clearly, $\psi_{\min}$ should be a good under-estimate of the expected cost per frame for the future unknown path. Therefore, the heuristic function $h(H_N^{t,T})$ can be derived as:

$$h(H_N^{t,T}) = (T-t)\psi_{\min} \tag{12.26}$$

Although $\psi_{\min}$ is obtained empirically, stack decoding based on Eq. (12.26) will generally find the optimal solution. However, the search using $\psi_{\min}$ usually runs very slowly, since Eq. (12.26) always under-estimates the true cost for any portion of speech. In practice, a heuristic function like ψ_{avg} that may over-estimate has to be used to prune more hypotheses. This speeds up the search at the expense of possible search errors, because ψ_{avg} should represent the average cost per frame for any portion of speech. In fact, there is an argument that one might be able to use a heuristic function even more than ψ_{avg}. The argument is that ψ_{avg} is derived from the correct path (training data) and the average cost per frame for all paths during search should be more than ψ_{avg} because the paths undoubtedly include correct and incorrect ones.

12.5.2. When to Extend New Words

Since stack decoding is executed asynchronously, it becomes necessary to detect when a phone/word ends, so that the search can extend to the next phone/word. If we have a cost measure that indicates how well an input feature vector of any length matches the evaluated model sequence, this cost measure should drop slowly for the correct phone/word and rise sharply for an incorrect phone/word. In order to do so, it implies we must be able to compare hypotheses of different lengths.

The first thing that comes to mind for this cost measure is simply the forward cost $-\log P(\mathbf{x}_1^t, s_t \mid w_1^k)$, which represents the likelihood of producing acoustic observation $\mathbf{x}_1^t$ based on word sequence w_1^k and ending at state s_t. However, it is definitely not suitable because it is deemed to be smaller for a shorter acoustic input vector. This causes the search to almost always prefer short phones/words, resulting in many insertion errors. Therefore, we must derive some normalized score that satisfies the desired property described above. The normalized cost $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$ can be represented as follows [6, 24]:

$$\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k) = -\log\left[\frac{P(\mathbf{x}_1^t, s_t \mid w_1^k)}{\gamma^t}\right] = -\log\left[P(\mathbf{x}_1^t, s_t \mid w_1^k)\right] + t\log\gamma \tag{12.27}$$

where γ $(0<\gamma<1)$ is a constant normalization factor.

Suppose the search is now evaluating a particular word w_k; we can define $\hat{C}_{\min}(t)$ as the minimum cost for $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$ for all the states of w_k, and $\alpha_{\max}(t)$ as the maximum forward probability for $P(\mathbf{x}_1^t, s_t \mid w_1^k)$ for all the states of w_k. That is,

$$\hat{C}_{\min}(t) = \min_{s_t \in w_k} \left[\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k) \right] \tag{12.28}$$

$$\alpha_{\max}(t) = \max_{s_t \in w_k} \left[P(\mathbf{x}_1^t \mid w_1^k, s_t) \right] \tag{12.29}$$

We want $\hat{C}_{\min}(t)$ to be near 0 just as long as the phone/word we are evaluating is the correct one and we have not gone beyond its end. On the other hand, if the phone/word we are evaluating is the incorrect one or we have already passed its end, we want the $\hat{C}_{\min}(t)$ to be rising sharply. Similar to the procedure of finding the admissible heuristic function, we can set the normalized factor γ empirically during training so that $\hat{C}_{\min}(T) = 0$ when we know the correct word sequence $\mathbf{W}$ that produces acoustic observation sequence $\mathbf{x}_1^T$. Based on Eq. (12.27), γ should be set to:

$$\gamma = \sqrt[T]{\alpha_{\max}(T)} \tag{12.30}$$

Figure 12.22 shows a plot of $\hat{C}_{\min}(t)$ as a function of time for correct match. In addition, the cost for the final state $FS(w_k)$ of word w_k, $\hat{C}(\mathbf{x}_1^t, s_t = FS(w_k) \mid w_1^k)$, which is the score for w_k-ending path, is also plotted. There should be a valley centered around 0 for $\hat{C}(\mathbf{x}_1^t, s_t = FS(w_k) \mid w_1^k)$, which indicates the region of possible ending time for the correct phone/word. Sometimes a stretch of acoustic observations may match better than the average cost, pushing the curve below 0. Similarly, a stretch of acoustic observations may match worse than the average cost, pushing the curve above 0.

There is an interesting connection between the normalized factor γ and the heuristic estimate of the expected cost per frame, ψ, defined in Eq. (12.25). Since the cost is simply the logarithm on the inverse posterior probability, we get the following equation:

$$\psi = \frac{-1}{T} \log P(\mathbf{x}_1^T \mid \hat{\mathbf{W}}) = -\log \left[\alpha_{\max}(T)^{1/T} \right] = -\log \gamma \tag{12.31}$$

Equation (12.31) reveals that these two quantities are basically the same estimate. In fact, if we subtract the heuristic function $f(H_N^t)$ defined in Eq. (12.24) by the constant $\log(\gamma^T)$, we get exactly the same quantity as the one defined in Eq. (12.27). Decisions on which path to extend first based on the heuristic function and when to extend the search to the next word/phone are basically centered on comparing partial theories with different lengths. Therefore, the normalized cost $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$ can be used for both purposes.

Based on the connection we have established, the heuristic function, $f(H_N^t)$, which estimates the goodness of a path is simply replaced by the normalized evaluation function $\hat{C}(\mathbf{x}_1^t, s_t \mid w_1^k)$. If we plot the un-normalized cost $C(\mathbf{x}_1^t, s_t \mid w_1^k)$ for the optimal path and other

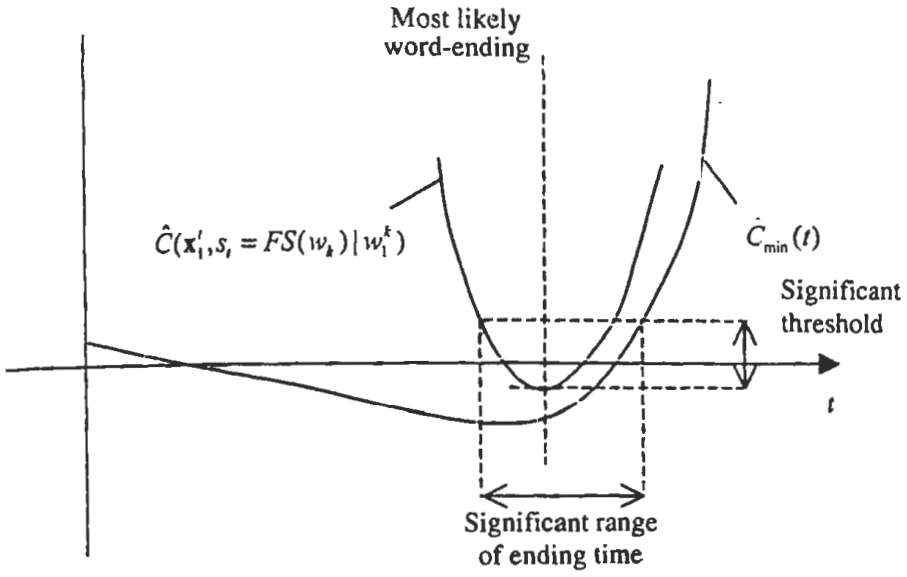


Figure 12.22 $\hat{C}_{\min}(t)$ and $\hat{C}(\mathbf{x}_1^t, s_t = FS(w_k) | w_1^k)$ as functions of time t. The valley region represents possible ending times for the correct phone/word.

competing paths as the function time t, the cost values increase as paths get longer (illustrated in Figure 12.23) because every frame adds some non-negative cost to the overall cost. It is clear that using un-normalized cost function $C(\mathbf{x}_1^t, s_t | w_1^k)$ generally results in a breadth-first search. What we want is an evaluation that decreases slightly along the optimal path, and hopefully increases along other competing paths. Clearly, the normalized cost function $\hat{C}(\mathbf{x}_1^t, s_t | w_1^k)$ fulfills this role, as shown in Figure 12.24.

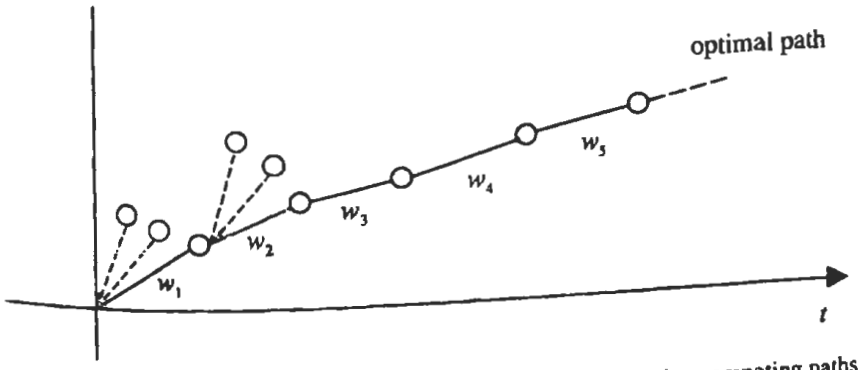


Figure 12.23 Unnormalized cost $C(\mathbf{x}_1^t, s_t | w_1^k)$ for optimal path and other competing paths as a function of time.

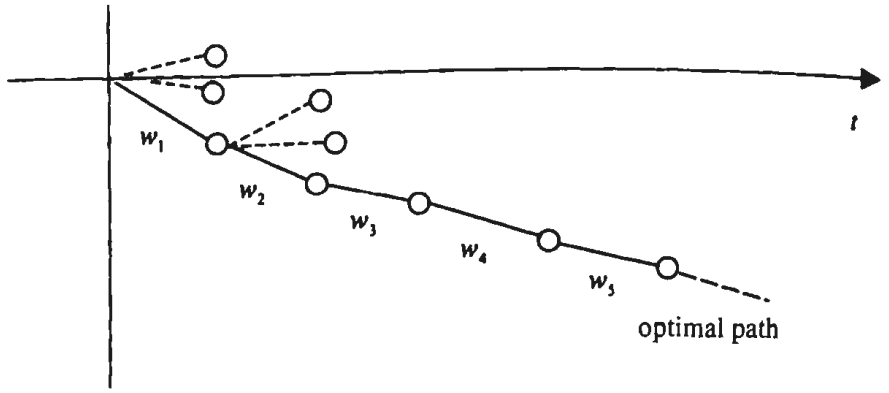


Figure 12.24 Normalized cost $\hat{C}(\mathbf{x}_1^t, s_i \mid w_1^k)$ for the optimal path and other competing paths as a function of time.

Equation (12.30) is a context-less estimation of the normalized factor, which is also referred to as zero-order estimate. To improve the accuracy of the estimate, you can use context-dependent higher-order estimates like [24]:

$$\gamma_i = \gamma(\mathbf{x}_i) \qquad \text{first-order estimate}$$
$$\gamma_i = \gamma(\mathbf{x}_i, \mathbf{x}_{i-1}) \qquad \text{second-order estimate}$$
$$\gamma_i = \gamma(\mathbf{x}_i, \mathbf{x}_{i-1}, \ldots, \mathbf{x}_{i-N+1}) \qquad n\text{-order estimate}$$

Since the normalized factor γ is estimated from the training data that is also used to train the parameters of the HMMs, the normalized factor γ_i tends to be an overestimate. As a result, $\alpha_{\max}(t)$ might rise slowly for test data even when the correct phone/word model is evaluated. This problem is alleviated by introducing some other scaling factor $\delta < 1$ so that $\alpha_{\max}(t)$ falls slowly for test data for when evaluating the correct phone/word model. The best solution for this problem is to use an independent data set other than the training data to derive the normalized factor γ_i.

12.5.3. Fast Match

Even with an efficient heuristic function and mechanism to determine the ending time for a phone/word, stack decoding could still be too slow for large-vocabulary speech recognition tasks. As described in Section 12.5.1, an effective underestimated heuristic function for the remaining portion of speech is very difficult to derive. On the other hand, a heuristic estimate for the immediate short segment that usually corresponds to the next phone or word may be feasible to attain. In this section, we describe the fast-match mechanism that reduces phone/word candidates for detailed match (expansion).

In asynchronous stack decoding, the most expensive step is to extend the best subpath. For a large-vocabulary search, it implies the calculation of $P(\mathbf{x}_t^{t+k} \mid w)$ over the entire vocabulary size $|V|$. It is desirable to have a fast computation to quickly reduce the possible

words starting at a given time t to reduce the search space. This process is often referred to as *fast match* [15, 35]. In fact, fast match is crucial to stack decoding, of which it becomes an integral part. Fast match is a method for the rapid computation of a list of candidates that constrain successive search phases. The expensive *detailed match* can then be performed after fast match. In this sense, fast match can be regarded as an additional pruning threshold to meet before new word/phone can be started.

Fast match, by definition, needs to use only a small amount of computation. However, it should also be accurate enough not to prune away any word/phone candidates that participate in the best path eventually. Fast match is, in general, characterized by the approximations that are made in the acoustic/language models in order to reduce computation. There is an obvious trade-off between these two objectives. Fortunately, many systems [15] have demonstrated that one needs to sacrifice very little accuracy in order to speed up the computation considerably.

Similar to *admissibility* in A* search, there is also an *admissibility* property in fast match. A fast match method is called admissible if it never prunes away the word/phone candidates that participate in the optimal path. In other words, a fast match is admissible if the recognition errors that appear in a system using the fast match followed by a detailed match are those that would appear if the detailed match were carried out for all words/phones in the vocabulary. Since fast match can be applied to either word or phone level, as we describe in the next section, we explain the admissibility for the case of word-level fast match for simplicity. The same principle can be easily extended to phone-level fast match.

Let V be the vocabulary and $C(\mathbf{X} \mid w)$ be the cost of a detailed match between input $\mathbf{X}$ and word w. Now $F(\mathbf{X} \mid w)$ is an estimator of $C(\mathbf{X} \mid w)$ that is accurate enough and fast to compute. A word list selected by fast match estimator can be attained by first computing $F(\mathbf{X} \mid w)$ for each word w of the vocabulary. Suppose w_b is the word for which the fast match has a minimum cost value:

$$w_b = \arg\min_{w \in V} F(\mathbf{X} \mid w) \tag{12.32}$$

After computing $C(\mathbf{X} \mid w_b)$, the detailed match cost for w_b, we form the fast match word list, Λ, from the word w in the vocabulary such that $F(\mathbf{X} \mid w)$ is no greater than $C(\mathbf{X} \mid w_b)$. In other words,

$$\Lambda = \{w \in V \mid F(\mathbf{X} \mid w) \le C(\mathbf{X} \mid w_b)\} \tag{12.33}$$

Similar to the admissibility condition for A* search [3, 33], the fast match estimator $F(\bullet)$ conducted in the way described above is admissible if and only if $F(\mathbf{X} \mid w)$ is always an under-estimator (lower bound) of detailed match $C(\mathbf{X} \mid w)$. That is,

$$F(\mathbf{X} \mid w) \le C(\mathbf{X} \mid w) \qquad \forall \mathbf{X}, w \tag{12.34}$$

The proof is straightforward. If the word w_c has a lower detailed match cost $C(\mathbf{X} \mid w_c)$, you can prove that it must be included in the fast match list Λ because

$$C(\mathbf{X} \mid w_c) \le C(\mathbf{X} \mid w_b) \text{ and } F(\mathbf{X} \mid w_c) \le C(\mathbf{X} \mid w_c) \Rightarrow F(\mathbf{X} \mid w_c) \le C(\mathbf{X} \mid w_b)$$

Therefore, based on the definition of Λ, $w_c \in \Lambda$.

Now the task is to find an admissible fast match estimator. Bahl et al. [6] proposed one fast match approximation for discrete HMMs. As we will see later, this fast match approximation is indeed equivalent to a simplification of the HMM structure. Given the HMM for word w and an input sequence x_1^T of codebook symbols describing the input signal, the probability that the HMM w produces the VQ sequence x_1^T is given by (according to Chapter 8):

$$P(x_1^T \mid w) = \sum_{s_1, s_2, \ldots s_T} \left[P_w(s_1, s_2, \ldots s_T) \prod_{i=1}^{T} P_w(x_i \mid s_i) \right] \tag{12.35}$$

Since we often use Viterbi approximation instead of the forward probability, the equation above can be approximated by:

$$P(x_1^T \mid w) \cong \max_{s_1, s_2 \ldots s_T} \left[P_w(s_1, s_2, \ldots s_T) \prod_{i=1}^{T} P_w(x_i \mid s_i) \right] \tag{12.36}$$

The detailed match cost $C(\mathbf{X} \mid w)$ can now be represented as:

$$C(\mathbf{X} \mid w) = \min_{s_1, s_2, \ldots s_T} \left\{ -\log \left[P_w(s_1, s_2, \ldots s_T) \prod_{i=1}^{T} P_w(x_i \mid s_i) \right] \right\} \tag{12.37}$$

Since the codebook size is finite, it is possible to compute, for each model w, the highest output probability for every VQ label c among all states s_k in HMM w. Let's define $m_w(c)$ to be the following:

$$m_w(c) = \max_{s_k \in w} P_w(c \mid s_k) = \max_{s_k \in w} b_k(c) \tag{12.38}$$

We can further define the $q_{\max}(w)$ as the maximum state sequence with respect to T, i.e., the maximum probability of any complete path in HMM w.

$$q_{\max}(w) = \max_{T} \left[P_w(s_1, s_2, \ldots s_T) \right] \tag{12.39}$$

Now let's define the fast match estimator $F(\mathbf{A} \mid w)$ as the following:

$$F(\mathbf{X} \mid w) = -\log \left[q_{\max}(w) \prod_{i=1}^{T} m_w(x_i) \right] \tag{12.40}$$

It is easy to show the fast match estimator $F(\mathbf{X} \mid w) \le C(\mathbf{X} \mid w)$ is admissible based on Eq. (12.38) to Eq. (12.40).

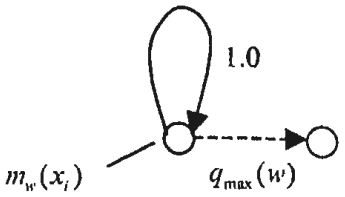


Figure 12.25 The equivalent one-state HMM corresponding to fast match computation defined in Eq. (12.40) [15].

The fast match estimator defined in Eq. (12.40) requires $T+1$ additions for a vector sequence of length T. The operation can be viewed as equivalent to the forward computation with a one-state HMM of the form shown in Figure 12.25. This correspondence can be interpreted as a simplification of the original multiple-state HMM into such a one-state HMM. It thus explains why fast match can be computed much faster than detailed match. Readers should note that this HMM is not actually a true HMM by strict definition, because the output probability distribution $m_w(c)$ and the transition probability distribution do not add up to one.

The fast match computation defined in Eq. (12.40) discards the sequence information with the model unit since the computation is independent of the order of input vectors. Therefore, one needs to decide the acoustic unit for fast match. In general, the longer the unit, the faster the computation is, and, therefore, the smaller the under-estimated cost $F(\mathbf{X}|w)$ is. It thus becomes a trade-off between accuracy and speed.

Now let's analyze the real speedup by using fast match to reduce the vocabulary V to the list Λ, followed by the detailed match. Let $|V|$ and $|\Lambda|$ be the sizes for the vocabulary V and the fast match short list Λ. Suppose t_f and t_d are the times required to compute one fast match score and one detailed match score for one word, respectively. Then, the total time required for the fast match followed by the detailed match is $t_f|V|+t_d|\Lambda|$, whereas the time required in doing the detailed match alone for the entire vocabulary is $t_d|V|$. The speed-up ratio is then given as follows:

$$\frac{1}{\left(\frac{t_f}{t_d}+\frac{|\Lambda|}{|V|}\right)} \tag{12.41}$$

We need t_f to be much smaller than t_d and $|\Lambda|$ to be much smaller than $|V|$ to have a significant speed-up using fast match. Using our admissible fast match estimator in Eq. (12.40), the time complexity of the computation for $F(\mathbf{X}|w)$ is T instead of N^2T for $C(\mathbf{X}|w)$, where N is the number of states in the detailed acoustic model. Therefore, the t_f/t_d saving is about N^2.

In general, in order to make $|\Lambda|$ much smaller than $|V|$, one needs a very accurate fast match estimator that could result in $t_f \approx t_d$. This is why we often relax the constraint of admissibility, although it is a nice principle to adhere to. In practice, most real-time speech recognition systems don't necessarily obey the admissibility principle with the fast match. For example, Bahl et al. [10], Laface et al., [22] and Roe et al., [36] used several techniques

to construct off-line groups of acoustically similar words. Armed with this grouping, they can use an aggressive fast match to select only a very short list of words, and words acoustically similar to the words in this list are added to form the short word list Λ for further detailed match processing. By doing so, they are able to report a very efficient fast match method that misses the correct word only 2% of the time. When non-admissible fast match is used, one needs to minimize the additional search error introduced by fast match empirically.

Bahl et al. [6] use a one-state HMM as their fast match units and a tree-structure lexicon similar to the lexical tree structures introduced in Chapter 13 to construct the short word list Λ for next-word expansion in stack decoding. Since the fast match tree search is also done in an asynchronous way, the ending time of each phone is detected using normalized scores similar to those described in Section 12.5.2. It is based on the same idea that this normalized score rises slowly for the correct phone, while it drops rapidly once the end of phone is encountered (so the model is starting to go toward the incorrect phones). During the asynchronous lexical tree search, the unpromising hypotheses are also pruned away by a pruning threshold that is constantly changing once a complete hypothesis (a leaf node) is obtained. On a 20,000-word dictation task, such a fast match scheme was about 100 times faster than detailed match and achieved real-time performance on a commercial workstation with only 0.34% increase in the word error rate being introduced by the fast match process.

12.5.4. Stack Pruning

Even with efficient heuristic functions, the mechanism to determine the ending time for phone/word, and fast match, stack decoding might still be too slow for large-vocabulary speech recognition tasks. A beam within the stack, which saves only a small number of promising hypotheses in the *OPEN* list, is often used to reduce search effort. This *stack pruning* is very similar to beam search. A predetermined threshold ε is used to eliminate hypotheses whose cost value is much worse than the best path so far.

Both fast match and stack pruning could introduce search errors where the eventual optimal path is thrown away prematurely. However, the impact could be reduced to a minimum by empirically adjusting the thresholds in both methods.

The implementation of stack decoding is, in general, more complicated, particularly when some inevitable pruning strategies are incorporated to make the search more efficient. The difficulty of devising both an effectively admissible heuristic function for $h(\bullet)$ and an effective estimation of normalization factors for boundary determination has limited the advantage that stack decoders have over Viterbi decoders. Unlike stack decoding, time-synchronous Viterbi beam search can use an easy comparison of same-length path without heuristic determination of word boundaries. As described in the earlier sections, these simple and unified features of Viterbi beam search allow researchers to incorporate various sound techniques to improve the efficiency of search. Therefore, time-synchronous Viterbi Beam search enjoys a much broader popularity in the speech community. However, the principle of stack decoding is essential particularly for n-best and lattice search. As we describe in Chapter 13, stack decoding plays a very crucial part in multiple-pass search strate-

gies for n-best and lattice search because the early pass is able to establish a near-perfect estimate of the remaining path.

12.5.5. Multistack Search

Even with the help of normalized factor γ or heuristic function $h(\bullet)$, it is still more effective to compare hypotheses of the same length than those of different lengths, because hypotheses with the same length are compared based on the true forward matching score. Inspired by the time-synchronous principle in Viterbi beam search, researchers [8, 35] propose a variant stack decoding based on multiple stacks.

Multistack search is equivalent to a best-first search algorithm running on multiple stacks time-synchronously. Basically, the search maintains a separate stack for each time frame t, so it never needs to compare hypotheses of different lengths. The search runs time-synchronously from left to right just like time-synchronous Viterbi search. For each time frame t, multistack search extracts the best path out of the t-stack, computes one-word extensions, and places all the new paths into the corresponding stacks. When the search finishes, the top path in the last stack is our optimal path. Algorithm 12.7 illustrates the multistack search algorithm.

This time-synchronous multistack search is designed based on the fact that by the time the t^{th} stack is extended, it already contains the best paths that could ever be placed into it. This phenomenon is virtually a variant of the dynamic programming principle introduced in Chapter 8. To make multistack more efficient, some heuristic pruning can be applied to reduce the computation. For example, when the top path of each stack is extended for one more word, we could only consider extensions between minimum and maximum duration. On the other hand, when some heuristic pruning is integrated into the multistack search, one might need to use a small beam in Step 2 of Algorithm 12.7 to extend more than just the best path to guarantee the admissibility.

ALGORITHM 12.7: *MULTISTACK SEARCH*

Step 1: Initialization: for each word v in vocabulary V

for $t = 1, 2, \ldots, T$

Compute $C(\mathbf{x}_1^t \mid v)$ and insert it to t^{th} stack

Step 2: Iteration: for $t = 1, 2, \ldots, T-1$

Sort the t^{th} stack and pop the top path $C(\mathbf{x}_1^t \mid w_1^k)$ out of the stack

for each word v in vocabulary V

for $\tau = t+1, t+2, \ldots, T$

Extend the path $C(\mathbf{x}_1^t \mid w_1^k)$ by word v to get $C(\mathbf{x}_1^\tau \mid w_1^{k+1})$

where $w_1^{k+1} = w_1^k \,\|\, v$ and $\|$ means string concatenation

Place $C(\mathbf{x}_1^\tau \mid w_1^{k+1})$ in τ^{th} stack

Step 3: Termination: Sort the T^{th} stack and the top path is the optimal word sequence

12.6. HISTORICAL PERSPECTIVE AND FURTHER READING

Search has been one of the most important topics in artificial intelligence since the origins of the field. It plays the central role in general problem solving [29] and computer games. [43], Nilsson's *Principles of Artificial Intelligence* [32] and Barr and Feigenbaum's *The Handbook of Artificial Intelligence* [11] contain a comprehensive introduction to state-space search algorithms. A* search was first proposed by Hart et al. [17]. A* was thought to be derived from Dijkstra's algorithm [13] and Moore's algorithm [27]. A* search is similar to the *branch-and-bound* algorithm [23, 39], widely used in *operations research*. The proof of admissibility of A* search can be found in [32].

The application of *beam search* in speech recognition was first introduced by the HARPY system [26]. It wasn't widely popular until BBN used it for their BYBLOS system [37]. There are some excellent papers with detailed description of the use of time-synchronous Viterbi beam search for continuous speech recognition [24, 31]. Over the years, many efficient implementations and improvements have been introduced for time-synchronous Viterbi beam search, so real-time large-vocabulary continuous speech recognition can be realized on a general-purpose personal computer.

On the other hand, stack decoding was first developed by IBM [9]. It is successfully used in IBM's large-vocabulary continuous speech recognition systems [3, 16]. Lacking a time-synchronous framework, comparing theories of different lengths and extending theories are more complex as described in this chapter. Because of the complexity of stack decoding, far fewer publications and systems are based on it than on Viterbi beam search [16, 19, 20, 35]. With the introduction of multistack search [8], stack decoding in essence has actually come very close to time-synchronous Viterbi beam search.

Stack decoding is typically integrated with fast match methods to improve its efficiency. Fast match was first implemented for isolated word recognition to obtain a list of potential word candidates [5, 7]. The paper by Gopalakrishnan et al. [15] contains a comprehensive description of fast match techniques to reduce the word expansion for stack decoding. Besides the fast match techniques described in this chapter, there are a number of alternative approaches [5, 21, 41]. Waast's fast match [41], for example, is based on a binary classification tree built automatically from data that comprise both phonetic transcription and acoustic sequence.

REFERENCES

[1] Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, 1974, Addison-Wesley Publishing Company.

[2] Alleva, F., X. Huang, and M. Hwang, "An Improved Search Algorithm for Continuous Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 307-310.

[3] Bahl, L.R. and et. al., "Large Vocabulary Natural Language Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, pp. 465-467.

[4] Bahl, L.R., *et al.*, "Language-Model/Acoustic Channel Balance Mechanism," *IBM Technical Disclosure Bulletin*, 1980, 23(7B), pp. 3464-3465.

[5] Bahl, L.R., *et al.*, "Obtaining Candidate Words by Polling in a Large Vocabulary Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1988, pp. 489-492.

[6] Bahl, L.R., *et al.*, "A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1993(1), pp. 59-67.

[7] Bahl, L.R., *et al.*, "Matrix Fast Match: a Fast Method for Identifying a Short List of Candidate Words for Decoding," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, pp. 345-347.

[8] Bahl, L.R., P.S. Gopalakrishnan, and R.L. Mercer, "Search Issues in Large Vocabulary Speech Recognition," *Proc. of the 1993 IEEE Workshop on Automatic Speech Recognition*, 1993, Snowbird, UT.

[9] Bahl, L.R., F. Jelinek, and R. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1983(2), pp. 179-190.

[10] Bahl, L.R., *et al.*, "Constructing Candidate Word Lists Using Acoustically Similar Word Groups," *IEEE Trans. on Signal Processing*, 1992(1), pp. 2814-2816.

[11] Barr, A. and E. Feigenbaum, *The Handbook of Artificial Intelligence: Volume I*, 1981, Addison-Wesley.

[12] Cettolo, M., R. Gretter, and R.D. Mori, "Knowledge Integration" in *Spoken Dialogues with Computers*, R.D. Mori, Editor 1998, London, Academic Press, pp. 231-256.

[13] Dijkstra, E.W., "A Note on Two Problems in Connection with Graphs," *Numerische Mathematik*, 1959, 1, pp. 269-271.

[14] Gauvain, J.L., *et al.*, "The LIMSI Speech Dictation System: Evaluation on the ARPA Wall Street Journal Corpus," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 129-132.

[15] Gopalakrishnan, P.S. 2and L.R. Bahl, "Fast Match Techniques," in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Norwell, MA, Kluwer Academic Publishers, pp. 413-428.

[16] Gopalakrishnan, P.S., L.R. Bahl, and R.L. Mercer, "A Tree Search Strategy for Large-Vocabulary Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 572-575.

[17] Hart, P.E., N.J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. on Systems Science and Cybernetics*, 1968, 4(2), pp. 100-107.

[18] Huang, X., *et al.*, "Microsoft Windows Highly Intelligent Speech Recognizer: Whisper," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, pp. 93-96.

[19] Jelinek, F., *Statistical Methods for Speech Recognition*, 1998, Cambridge, MA, MIT Press.

[20] Kenny, P., *et al.*, "A*—Admissible Heuristics for Rapid Lexical Access," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**, pp. 49-58.

[21] Kenny, P., *et al.*, "A New Fast Match for Very Large Vocabulary Continuous Speech Recognition," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 656-659.

[22] Laface, P., L. Fissore, and F. Ravera, "Automatic Generation of Words toward Flexible Vocabulary Isolated Word Recognition," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 2215-2218.

[23] Lawler, E.W. and D.E. Wood, "Branch-and-Bound Methods: A Survey," *Operations Research*, 1966(14), pp. 699-719.

[24] Lee, K.F. and F.A. Alleva, "Continuous Speech Recognition" in *Recent Progress in Speech Signal Processing*, S. Furui and M. Sondhi, eds., 1990, Marcel Dekker, Inc.

[25] Lee, K.F., H.W. Hon, and R. Reddy, "An Overview of the SPHINX Speech Recognition System," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1990, 38(1), pp. 35-45.

[26] Lowerre, B.T., *The HARPY Speech Recognition System*, PhD Thesis in Computer Science Department, 1976, Carnegie Mellon University.

[27] Moore, E.F., "The Shortest Path Through a Maze," *Int. Symp. on the Theory of Switching*, 1959, Cambridge, MA, Harvard University Press, pp. 285-292.

[28] Murveit, H., *et al.*, "Large Vocabulary Dictation Using SRI's DECIPHER Speech Recognition System: Progressive Search Techniques," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 319-322.

[29] Newell, A. and H.A. Simon, *Human Problem Solving*, 1972, Englewood Cliffs, NJ, Prentice Hall.

[30] Ney, H. and X. Aubert, "Dynamic Programming Search: From Digit Strings to Large Vocabulary Word Graphs," in *Automatic Speech and Speaker Recognition*, C.H. Lee, F. Soong and K.K. Paliwal, eds., 1996, Boston, Kluwer Academic Publishers, pp. 385-412.

[31] Ney, H. and S. Ortmanns, *Dynamic Programming Search for Continuous Speech Recognition*, in *IEEE Signal Processing Magazine*, 1999, pp. 64-83.

[32] Nilsson, N.J., *Principles of Artificial Intelligence*, 1982, Berlin, Germany, Springer Verlag.

[33] Nilsson, N.J., *Artificial Intelligence: A New Synthesis*, 1998, Academic Press/Morgan Kaufmann.

[34] Normandin, Y., R. Cardin, and R.D. Mori, "High-Performance Connected Digit Recognition Using Maximum Mutual Information Estimation," *IEEE Trans. on Speech and Audio Processing*, 1994, 2(2), pp. 299-311.

[35] Paul, D.B., "An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, California, pp. 25-28.

[36] Roe, D.B. and M.D. Riley, "Prediction of Word Confusabilities for Speech Recognition," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 227-230.

[37] Schwartz, R., *et al.*, "Context-Dependent Modeling for Acoustic-Phonetic Recognition of Speech Signals," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1985, Tampa, FLA, pp. 1205-1208.

[38] Steinbiss, V., *et al.*, "The Philips Research System for Large-Vocabulary Continuous-Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1993, Berlin, Germany, pp. 2125-2128.

[39] Taha, H.A., *Operations Research: An Introduction*, 6th ed, 1996, Prentice Hall.

[40] Tanimoto, S.L., *The Elements of Artificial Intelligence : An Introduction Using Lisp*, 1987, Computer Science Press, Inc.

[41] Waast, C. and L.R. Bahl, "Fast Match Based on Decision Tree," *Proc. of the European Conf. on Speech Communication and Technology*, 1995, Madrid, Spain, pp. 909-912.

[42] Winston, P.H., *Artificial Intelligence*, 1984, Reading, MA, Addison-Wesley.

[43] Winston, P.H., *Artificial Intelligence*, 3rd ed, 1992, Reading, MA, Addison-Wesley.

[44] Woodland, P.C., *et al.*, "Large Vocabulary Continuous Speech Recognition Using HTK," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 125-128.

CHAPTER 13

Large-Vocabulary Search Algorithms

Chapter 12 discussed the basic search techniques for speech recognition. However, the search complexity for large-vocabulary speech recognition with high-order language models is still difficult to handle. In this chapter we describe efficient search techniques in the context of time-synchronous Viterbi beam search, which becomes the choice for most speech recognition systems because it is very efficient. We use Microsoft Whisper as our case study to illustrate the effectiveness of various search techniques. Most of the techniques discussed here can also be applied to stack decoding.

With the help of beam search, it is unnecessary to explore the entire search space or the entire trellis. Instead, only the promising search state-space needs to be explored. Please keep in mind the distinction between the implicit search graph specified by the grammar network and the explicit partial search graph that is actually constructed by the Viterbi beam search algorithm.

In this chapter we first introduce the most critical search organization for large-vocabulary speech recognition—tree lexicons. Tree lexicons significantly reduce potential search space, although they introduce many practical problems. In particular, we need to

address problems such as reentrant lexical trees, factored language model probabilities, subtree optimization, and subtree polymorphism.

Various other efficient techniques also are introduced. Most of these techniques aim for clever pruning with the hope of sparing the correct paths. For more effective pruning, different layers of beams are usually used. While fast match techniques described in Chapter 12 are typically required for stack decoding, similar concepts and techniques can be applied to Viterbi beam search. In practice, the look-ahead strategy is equally effective for Viterbi beam search.

Although it is always desirable to use all the knowledge sources (KSs) in the search algorithm, some are difficult to integrate into the left-to-right time-synchronous search framework. One alternative strategy is to first produce an ordered list of sentence hypotheses (a.k.a. *n-best list*), or a lattice of word hypotheses (a.k.a. *word lattice*) using relatively inexpensive KSs. More expensive KSs can be used to rescore the n-best list or the word lattice to obtain the refined result. Such a multipass strategy has been explored in many large-vocabulary speech recognition systems. Various algorithms to generate sufficient n-best lists or the word lattices are described in the section on multipass search strategies.

Most of the techniques described in this chapter rely on nonadmissible heuristics. Thus, it is critical to derive a framework to evaluate different search strategies and pruning parameters.

13.1. EFFICIENT MANIPULATION OF A TREE LEXICON

The lexicon entry is the most critical component for large-vocabulary speech recognition, since the search space grows linearly along with increased linear vocabulary. Thus an efficient framework for handling large vocabulary undoubtedly becomes the most critical issue for efficient search performance.

13.1.1. Lexical Tree

The search space for n-gram discussed in Chapter 12 is organized based on a straightforward linear lexicon, i.e., each word is represented as a linear sequence of phonemes, independent of other words. For example, the phonetic similarity between the words *task* and *tasks* is not leveraged. In a large-vocabulary system, many words may share the same beginning phonemes. A tree structure is a natural representation for a large-vocabulary lexicon, as many phonemes can be shared to eliminate redundant acoustic evaluations. The lexical tree-based search is thus essential for building a real-time[1] large-vocabulary speech recognizer.

[1] The term *real-time* means the decoding process takes no longer than the duration of the speech. Since the decoding process can take place as soon as the speech starts, such a real-time decoder can provide real instantaneous responses after speakers finish talking.

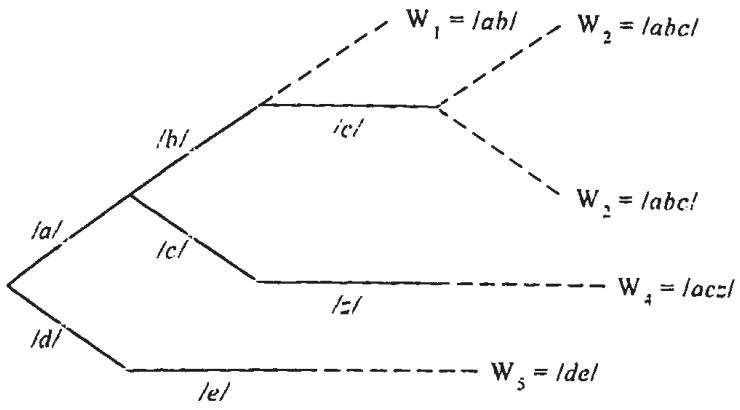


Figure 13.1 An example of a lexical tree, where each branch corresponds to a shared phoneme and the leaf corresponds to a word.

Figure 13.1 shows an example of such a lexical tree, where common beginning phonemes are shared. Each leaf corresponds to a word in the vocabulary. Please note that an extra null arc is used to form the leaf node for each word. This null arc has the following two functions:

1. When the pronunciation transcription of a word is a prefix of other ones, the null arc can function as one branch to end the word.
2. When there are homophones in the lexicon, the null arcs can function as linguistic branches to represent different words such as *two* and *to*.

The advantage of using such a lexical tree representation is obvious: it can effectively reduce the state search space of the trellis. Ney et al. [32] reported that a lexical tree representation of a 12,306-word lexicon with only 43,000 phoneme arcs had a saving of a factor of 2.5 over the linear lexicon with 100,800 phoneme arcs. Lexical trees are also referred to as *prefix trees*, since they are efficient representations of lexicons with sharing among lexical entries that have a common prefix. Table 13.1 shows the distribution of phoneme arcs for this 12,306-word lexical tree. As one can see, even in the fifth level the number of phoneme arcs is only about one-third of the total number of words in the lexicon.

Table 13.1 Distribution of the tree phoneme arcs and active tree phoneme arc for a 12,306-word lexicon using a lexical tree representation [32].

Level	1	2	3	4	5	6	≥7
Phoneme arcs	28	331	1511	3116	4380	4950	29.200
Average active arcs	23	233	485	470	329	178	206

The saving by using a lexical tree is substantial, because it not only results in considerable memory saving for representing state-search space but also saves tremendous time by searching far fewer potential paths. Ney et al. [32] report that a tree organization of the lexicon reduces the total search effort by a factor of 7 over the linear lexicon organization. This is because the lion's share of hypotheses during a typical large-vocabulary search is on the first and second phonemes of a word. Haeb-Umbach et al. [23] report that for a 12,306-word dictation task, 79% and 16% of the state hypotheses are in the first and second phonemes, when analyzing the distribution of the state hypotheses over the state position within a word. Obviously, the effect is caused by the ambiguities at the word boundaries. The lexical tree representation reduces that effort by evaluating common phonetic prefixes only once. Table 13.1 also shows the average number of active phoneme arcs in the layers of the lexical tree [32]. Based on this table, you can expect that the overall search cost is far less than the size of the vocabulary. This is the key reason why lexical tree search is widely used for large-vocabulary continuous speech recognition systems.

The lexical tree search requires a sophisticated implementation because of a fundamental deficiency—*a branch in a lexical tree representation does not correspond to a single word with the exception of branches ending in a leaf.* This deficiency translates to the fact that a unique word identity is not determined until a leaf of the tree is reached. This means that any decision about the word identity needs to be delayed until the leaf node is reached, which results in the following complexities.

- Unlike a linear lexicon, where the language model score can be applied when starting the acoustic search of a new word, the lexical tree representation has to delay the application of the language model probability until the leaf is reached. This may result in an increased search effort, because the pruning needs to be done on a less reliable measure, unless a factored language model is used, as discussed in Section 13.1.3.
- Because of the delay of language model contribution by one word, we need to keep a separate copy of an entire lexical tree for each unique language model history.

13.1.2. Multiple Copies of Pronunciation Trees

A simple lexical tree is sufficient if no language model or a unigram is used. This is because the decision at time t depends on the current word only. However, for higher-order n-gram models, the linguistic state cannot be determined locally. A tree copy is required for each language model state. For bigrams, a tree copy is required for each predecessor word. This may seem to be astonishing, because the potential search space is increased by the vocabulary size. Fortunately, experimental results show only a small number of tree copies are required, because efficient pruning can eliminate most of the unneeded ones. Ney et al. [32] report that the search effort using bigrams is increased by only a factor of 2 over the unigram

case. In general, when more detailed (better) acoustic and/or language models are used, the effect of a potentially increased search space is often compensated by a more focused beam search from the use of more accurate models. In other words, although the static search space might increase significantly by using more accurate models, the dynamic search space can be under control (sometimes even smaller), thanks to improved evaluation functions.

To deal with tree copies [19, 23, 37], you can create redundant subtrees. When copies of lexical trees are used to disambiguate active linguistic contexts, many of the active state hypotheses correspond to the same redundant unigram state, due to the postponed application of language models. To apply the language model sooner, and to eliminate redundant unigram state computations, a successor tree, T_i, can be created for each linguistic context i. T_i encodes the nonzero n-grams of the linguistic context i as an isomorphic subgraph of the unigram tree, T_0. Figure 13.2 shows the organization of such successor trees and unigram tree for bigram search. For each word w a successor tree, T_w is created with the set of successor words that have nonzero bigram probabilities. Suppose u is a successor of w; the bigram probability $P(u \mid w)$ is attached to the transition connecting the leaf corresponding to u in the successor tree T_w, with the root of the successor tree T_u. The unigram tree is a full-size lexical tree and is shared by all words as the back-off lexical tree. Each leaf of the unigram tree corresponds to one of $|V|$ words in the vocabulary and is linked to the root of its bigram successor tree (T_u) by an arc with the corresponding unigram probability $P(u)$. The backoff weight, $\alpha(u)$, of predecessor u is attached to the arc which links the root of successor tree T_u to the root of the unigram tree.

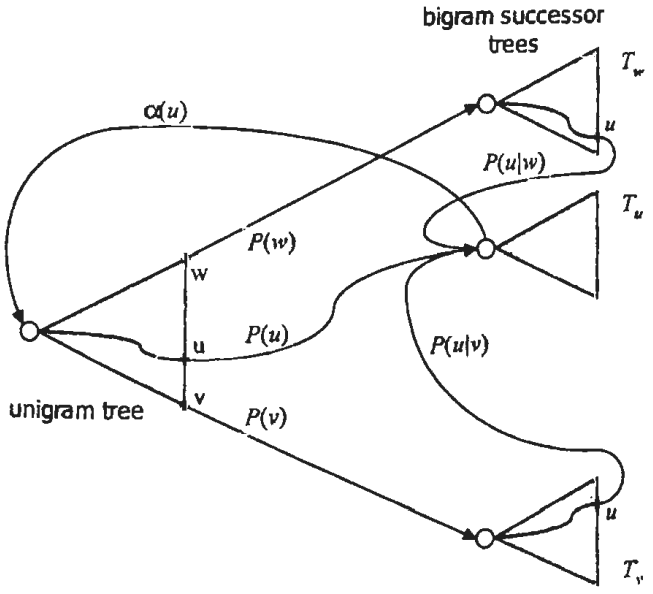


Figure 13.2 Successor trees and unigram trees for bigram search [13].

A careful search organization is required to avoid computational overhead and to guarantee a linear time complexity for exploring state hypotheses. In the following sections we describe techniques to achieve efficient lexical tree recognizers. These techniques include factorization of language model probabilities, tree optimization, and exploiting subtree dominance.

13.1.3. Factored Language Probabilities

As mentioned in Section 13.1.2, search is more efficient if a detailed knowledge source can be applied at an early stage. The idea of *factoring* the language model probabilities across the tree is one such example [4, 19]. When more than one word shares a phoneme arc, the upper bound of their probability can be associated to that arc.[2] The factorization can be applied to both the full lexical tree (unigram) and successor trees (bigram or other higher-order language models).

An unfactored tree only has language model probabilities attached to the leaf nodes, and all the internal nodes have probability 1.0. The procedure for factoring the probabilities across the tree computes the maximum of each node n in the tree according to Eq. (13.1). The tree can then be factored according to Eq. (13.2) so when you traverse the tree you can multiply $F^*(n)$ along the path to get the needed language probability.

$$P^*(n) = \max_{x \in child(n)} P(x) \tag{13.1}$$

$$F^*(n) = \frac{P^*(n)}{P^*(parent(n))} \tag{13.2}$$

An illustration of the factored probabilities is shown in Table 13.2. Using this lexicon, we create the tree depicted in Figure 13.3(a). In this figure the unlabeled internal nodes have a probability of 1.0. We distribute the probabilities according to Eq. (13.1) in Figure 13.3(b), which is factored according to Eq. (13.2), resulting in Figure 13.3(c).

Table 13.2 Sample probabilities $P(w_j)$ and their pseudoword pronunciations [4].

w_j	**Pronunciation**	$P(w_j)$
w_0	/a b c/	0.1
w_1	/a b c/	0.4
w_2	/a c z/	0.3
w_3	/d e/	0.2

[2] The choice of upper bound is because it is an admissible estimate of the path no matter which word will be chosen later.

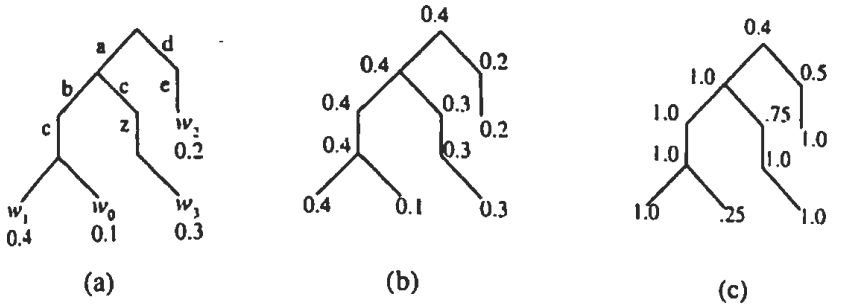


Figure 13.3 (a) Unfactored lexical tree; (b) distributed probabilities with computed $P^*(n)$; (c) factored tree $F^*(n)$ [4].

Using the upper bounds in the factoring algorithm is not an approximation, since the correct language model probabilities are calculated by the product of values traversed along each path from the root to the leaves. However, you should note that the probabilities of all the branches of a node do not sum to one. This can solved by replacing the upper-bound (max) function in Eq. (13.1) with the sum.

$$P^*(n) = \sum_{x \in child(n)} P(x) \tag{13.3}$$

To guarantee that all the branches sum to one, Eq. (13.2) should also be replaced by the following equation:

$$F^*(n) = \frac{P^*(n)}{\sum_{x \in child(parent(n))} P^*(x)} \tag{13.4}$$

A new illustration of the distribution of LM probabilities by using sum instead of upper bound is shown in Figure 13.4. Experimental results have shown that the factoring method with either sum or upper bound has comparable search performance.

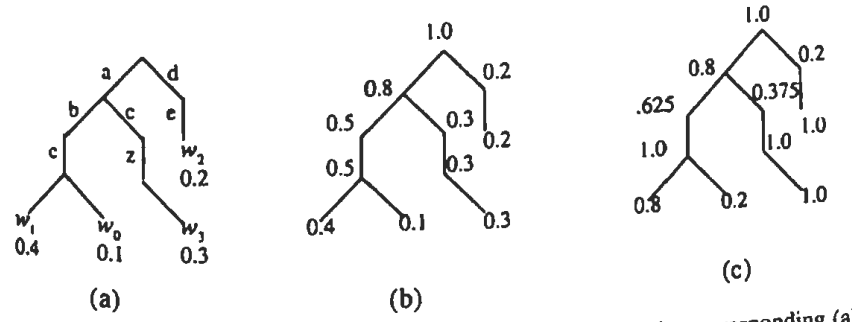


Figure 13.4 Using sum instead of upper bound when factoring tree, the corresponding (a) unfactored lexical tree; (b) distributed probabilities with computed $P^*(n)$; (c) factored tree with computed $F^*(n)$ [4].

One interesting observation is that the language model score can be regarded as a heuristic function to estimate the linguistic expectation of the current word to be searched. In a linear representation of the pronunciation lexicon, application of the linguistic expectation was straightforward, since each state is associated with a unique word. Therefore, given the context defined by the hypothesis under consideration, the expectation for the first phone of word w_l is just $P(w_l \mid w_1^{l-1})$. After the first phone, the expectation for the rest of the phones becomes 1.0, since there is only one possible phone sequence when searching the word w_l. However, for the tree lexicon, it is necessary to compute $E(p_j \mid p_1^{j-1}, w_i^{l-1})$, the expectation of phone p_j given the phonetic prefix p_1^{j-1} and the linguistic context w_1^{l-1}. Let $\phi(j, w_k)$ denote the phonetic prefix of length j for w_k. Based on Eqs. (13.1) and (13.2), we can compute the expectation as:

$$E(p_j \mid p_1^{j-1}, w_1^{i-1}) = \frac{P(w_c \mid w_1^{i-1})}{P(w_p \mid w_1^{i-1})} \tag{13.5}$$

where $c = \arg\max_k(w_k \mid w_1^{i-1}, \phi(j, w_k) = p_1^j)$ and $p = \arg\max_k(w_k \mid w_1^{i-1}, \phi(j-1, w_k) = p_1^{j-1})$. Based on Eq. (13.5), an arbitrary n-gram model or even a stochastic context-free grammar can be factored accordingly.

13.1.3.1. Efficient Memory Organization of Factored Lexical Trees

A major drawback to the use of successor trees is the large memory overhead required to store the additional information that encodes the structure of the tree and the factored linguistic probabilities. For example, the 5.02 million bigrams in the 1994 NABN (North American Business News) model require 18.2 million nodes. Given a compact binary tree representation that uses 4 bytes of memory per node, 72.8 million bytes are required to store the predecessor-dependent lexical trees. Furthermore, this tree representation is not as amenable to data compression techniques as the linear bigram representation.

The factored probability of successor trees can be encoded as efficiently as the n-gram model based on Algorithm 13.1, i.e., one n-gram record results in one constant-sized record. Step 3 is illustrated in Figure 13.5(b), where the heavy line ends at the most recently visited node that is not a direct ancestor. The encoding result is shown in Table 13.3.

ALGORITHM 13.1: *ENCODING THE LEXICAL SUCCESSOR TREES (LST)*

For each linguistic context:
Step 1: Distribute the probabilities according to Eq. (13.1).
Step 2: Factor the probabilities according to Eq. (13.2).
Step 3: Perform a depth-first traversal of the LST and encode each leaf record,
(a) the depth of the most recently visited node that is not a direct ancestor,
(b) the probability of the direct ancestor at the depth in (a),
(c) the word identity.

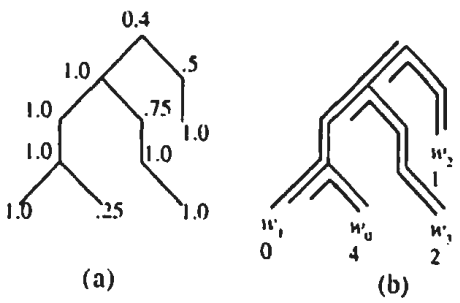


Figure 13.5 (a) Factored tree; (b) tree with common prefix-length annotation.

Clearly the new data structure meets the requirements set forth, and, in fact, it only requires additional $\log(n)$ bits per record (n is the depth of the tree). These bits encode the common prefix length for each word. Naturally this requires some modification to the decoding procedure. In particular, the decoder must scan a portion of the n-gram successor list in order to determine which tree nodes should be activated. Depending on the structure of the tree (which is determined by the acoustic model, the lexicon, and language model), the tree structure can be interpreted at runtime or cached for rapid access if memory is available.

Table 13.3 Encoded successor lexical tree; each record corresponds to one augmented factored n-gram.

w_j	Depth	$F^*(w_j)$
w_1	0	0.4
w_0	4	0.25
w_3	2	0.75
w_2	1	0.5

13.1.4. Optimization of Lexical Trees

We now investigate ways to handle the huge search network formed by the multiple copies of lexical trees in different linguistic contexts. The factorization of lexical trees actually makes it easier to search. First, after the factorization of the language model, the intertree transitions shown in Figure 13.2 no longer have the language model scores attached because they are already applied completely before leaving the leaves. Moreover, as illustrated in Figure 13.3, many transitions toward the end of a single-word path now have an associated transition probability that is equal to 1. This observation implies that there could be many duplicated subtrees in the network. Those duplicated subtrees can then be merged to save both space and computation by eliminating redundant (unnecessary) state evaluation. Unlike pruning, this saving is based on the dynamic programming principle, without introducing any potential error.

13.1.4.1. Optimization of Finite State Network

One way to compress the lexical tree network is to use a similar algorithm for optimizing the number of states in a deterministic finite state automaton. The optimization algorithm is based on the *indistinguishable* property of states in a finite state automaton. Suppose that s_1 and s_2 are the initial states for automata T_1 and T_2, then s_1 and s_2 are said to be *indistinguishable* if the languages accepted by automata T_1 and T_2 are exactly the same. If we consider our lexical tree network as a finite state automaton, the symbol emitted from the transition arc includes not only the phoneme identity, but also the factorized language model probability.

The general set-partitioning algorithm [1] can be used for the reduction of finite state automata. The algorithm starts with an initial partition of the automaton states and iteratively refines the partition so that two states s_1 and s_2 are put in the same block B_i if and only if $f(s_1)$ and $f(s_2)$ are both in the same block B_j. For our purpose, $f(s_1)$ and $f(s_2)$ can be defined as the destination state given a phone symbol (in the factored trees, the pair <*phone, LM-probability*> can be used). Each time a block is partitioned, the smaller subblock is used for further partitioning. The algorithm stops when all the states that transit to some state in a particular block with arcs labeled with the same symbol are in the same block. When the algorithm halts, each block of the resulting partition is composed of *indistinguishable* states, and those states within each block can then be merged. The algorithm is guaranteed to find the automaton with the minimum number of states. The algorithm has a time complexity of $O(MN \log N)$, where M is the maximum number of branching (fan-out) factors in the lexical tree and N is the number of states in the original tree network.

Although the above algorithm can give optimal finite state networks in terms of number of states, such an optimized network may be difficult to maintain, because the original lexical tree structure could be destroyed and it may be troublesome to add any new word into the tree network [1].

13.1.4.2. Subtree Isomorphism

The finite state optimization algorithm described above does not take advantage of the tree structure of the finite state network, though it generates a network with a minimum number of states. Since our finite state network is a network of trees, the indistinguishability property is actually the same as the definition of subtree isomorphism. Two subtrees are said to be *isomorphic* to each other if they can be made equivalent by permuting the successors. It should be straightforward to prove that two states are indistinguishable, if and only if their subtrees are isomorphic.

There are efficient algorithms [1] to detect whether two subtrees are isomorphic. For all possible pairs of states u and v, if the subtrees starting at u and v, $ST(u)$ and $ST(v)$, are isomorphic, v is merged into u and $ST(v)$ can be eliminated. Note that only internal nodes need to be considered for subtree isomorphism check. The time complexity for this algorithm is $O(N^2)$ [1].

13.1.4.3. Sharing Tails

A *linear tail* in a lexical tree is defined as a subpath ending in a leaf and going through states with a unique successor. It is often referred as a *single-word subpath*. It can be proved that such a linear tail has unit probability attached to its arcs according to Eqs. (13.1) and (13.2). This is because LM probability factorization *pushes forward* the LM probability attached to the last arc of the linear tail, leaving arcs with unit probability. Since all the tails corresponding to the same word w in different successor trees are linked to the root of successor tree T_w,[1] the subtree starting from the first state of each linear tail is isomorphic to the subtree starting from one of the states forming the longest linear tail of w. A simple algorithm to take advantage of this share-tail topology can be employed to reduce the lexical tree network.

Figure 13.6 and Figure 13.7 show a lexical tree network before and after shared-tail optimization. For each word, only the longest linear tail is kept. All other tails can be removed by linking them to an appropriate state in the longest tail, as shown in Figure 13.7.

Shared-tail optimization is not global optimization, because it considers only some special topology optimization. However, there are some advantages associated with shared-tail optimization. First, in practice, duplicated linear tails account for most of the redundancy in lexical tree networks [12]. Moreover, shared-tail optimization has a nice property of maintaining the basic lexical tree structure for the optimized tree network.

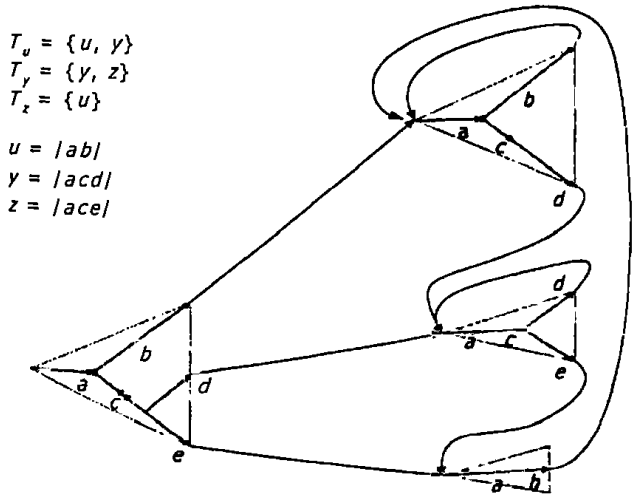


Figure 13.6 An example of a lexical tree network without shared-tail optimization [12]. The vocabulary includes three words, u, y, and z. T_u, T_y, and T_z are the successor trees for u, y, and z respectively [13].

[1] We assume bigram is used in the discussion of "sharing tails."

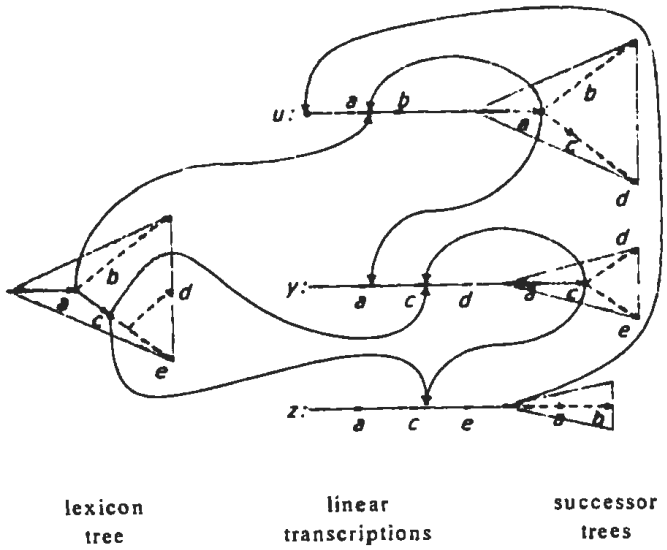


Figure 13.7 The lexical tree network in Figure 13.6 after shared-tail optimization [12].

13.1.5. Exploiting Subtree Polymorphism

The techniques of optimizing the network of successor lexical trees can only eliminate identical subtrees in the network. However, there are still many subtrees that have the same nodes and topology but with different language model scores attached to the arcs. The acoustic evaluation for those subtrees is unnecessarily duplicated. In this section we exploit *subtree dominance* for additional saving.

A subtree instance is *dominated* when the best outcome in that subtree is not better than the worst outcome in another instance of that subtree. The evaluation becomes redundant for the dominated subtree instance. Subtree isomorphism and shared-tail are cases of subtree dominance, but they require prearrangement of the lexical tree network as described in the previous section.

If we need to implement lexical tree search dynamically, the network optimization algorithms are not suitable. Although subtree dominance can be computed using minimax search [35] during runtime, this requires that information regarding subtree isomorphism be available for all corresponding pairs of states for each successor tree T_w. Unfortunately, it is not practical in terms of either computation or space.

In place of computing strict subtree dominance, a *polymorphic* linguistic context assignment to reduce redundancy is employed by estimating subtree dominance based on local information and ignoring the subgraph isomorphism problem. Polymorphic context assignment involves keeping a single copy of the lexical tree and allowing each state to assume the linguistic context of the most promising history. The advantage of this approach is that it employs maximum sharing of data structures and information, so each node in the tree is

evaluated, at most, once. However, the use of local knowledge to determine the dominant context could introduce significant errors because of premature pruning. Whisper [4] reports a 65.7% increase in error rate when only the dominant context is kept, based on local knowledge.

To recover the errors created by using local linguistic information to estimate subtree dominance, you need to delay the decision regarding which linguistic context is most promising. This can be done by keeping a heap of contexts at each node in the tree. The heap maintains all contexts (linguistic paths) whose probabilities are within a constant threshold ε, of that of the best global path. The effect of the ε-heap is that more contexts are retained for high-probability states in the lexical tree. The pseudocode fragment in Algorithm 13.2 [3] illustrates a transition from state s_n in context c to state s_m. The terminology used in Algorithm 13.2 is listed as follows:

- $(-\log P(s_m \mid s_n, c))$ is the cost associated with applying acoustic model matching and language model probability of state s_m transited from s_n in context c.
- $InHeap(s_m, c)$ is true if context c is in the heap corresponding to state s_m.
- $Cost(s_m, c)$ is the cost for context c in state s_m.
- $StateInfo(s_m, c)$ is the auxiliary state information associated with context c in state s_m.
- $Add(s_m, c)$ adds context c to the state s_m heap.
- $Delete(s_m, c)$ deletes context c from state s_m heap.
- $WorstContext(s_m)$ retrieves the worst context from the heap of state s_m.

ALGORITHM 13.2: *HANDLING MULTIPLE LINGUISTIC CONTEXTS IN A LEXICAL TREE*

```
1. d = Cost(s_n, c) + (-log P(s_m | s_n, c))
2. If InHeap(s_m, c) then
     If d < Cost(s_m, c) then
          Cost(s_m, c) = d
          StateInfo(s_m, c) = StateInfo(s_n, c)
   else if d < BestCost(s_m) + ε then
     Add(s_m, c); StateInfo(s_m, c) = StateInfo(s_n, c)
     Cost(s_m, c) = d
     else
     w = WorstContext(s_m)
     If d < Cost(s_m, w) then
          Delete(s_m, w)
          Add(s_m, c); StateInfo(s_m, c) = StateInfo(s_n, c)
          Cost(s_m, c) = d
```

When higher-order n-gram is used for lexical tree search, the potential heap size for lexical tree nodes (some also refer to *prefix nodes*) could be unmanageable. With decent acoustic models and efficient pruning, as illustrated in Algorithm 13.2, the average heap size for active nodes in the lexical tree is actually very modest. For example, Whisper's average heap size for active nodes in the 20,000-word WSJ lexical tree decoder is only about 1.6 [3].

13.1.6. Context-Dependent Units and Inter-Word Triphones

So far, we have implicitly assumed that context-independent models are used in the lexical tree search. When context-dependent phonetic or subphonetic models, as discussed in Chapter 9, are used for better acoustic models, the construction and use of a lexical tree become more complicated.

Since senones represent both subphonetic and context-dependent acoustic models, this presents additional difficulty for use in lexical trees. Let's assume that a three-state context-dependent HMM is formed from three senones, one for each state. Each senone is context-dependent and can be shared by different allophones. If we use allophones as the units for lexical tree, the sharing may be poor and fan-out unmanageable. Fortunately, each HMM is uniquely identified by the sequence of senones used to form the HMM. In this way, different context-dependent allophones that share the same *senone sequence* can be treated as the same. This is especially important for lexical tree search, since it reduces the order of the fan-out in the tree.

Interword triphones that require significant fan-ins for the first phone of a word and fan-outs for the last phones usually present an implementation challenge for large-vocabulary speech recognition. A common approach is to delay full interword modeling until a subsequent rescoring phase.[4] Given a sufficiently rich lattice or word graph, this is a reasonable approach, because the static state space in the successive search has been reduced significantly. However, as pointed out in Section 13.1.2, the size of the dynamic state space can remain under control when detailed models are used to allow effective pruning. In addition, a multipass search requires an augmented set of acoustic models to effectively model the biphone contexts used at word boundaries for the first pass. Therefore, it might be desirable to use genuine interword acoustic models in the single-pass search.

Instead of expanding all the fan-ins and fan-outs for inter-word context-dependent phone units in the lexical tree, three *metaunits* are created.

1. The first metaunit, which has a known right context corresponding to the second phone in the word, but uses open left context for the first phone of a word (sometimes referred to as the *word-initial unit*). In this way, the fan-in is represented as a subgraph shared by all words with the same initial left-context-dependent phone.

[4] Multipass search strategy is described in Section 13.3.5.

2. Another metaunit, which has a known left context corresponding to the second-to-last phone of the word, but uses open right context for the last phone of a word (sometimes referred to as the *word-final unit*). Again, the fan-out is represented as a subgraph shared by all words with the same final right-context-dependent phone.
3. The third metaunit, which has both open left and right contexts, and is used for single-phone word unit.

By using these metaunits we can keep the states for the lexical trees under control, because the fan-in and fan-out are now represented as a single node.

During recognition, different left or right contexts within the same metaunit are handled using Algorithm 13.2, where the different acoustic contexts are treated similarly as different linguistic contexts. The open left-context metaunit (fan-ins) can be dealt with in a straightforward way using Aglorithm 13.2, because the left context is always known (the last phone of the previous word) when it is initiated. On the other hand, the open right-context metaunit (fan-out) needs to explore all possible right contexts because the next word is not known yet. To reduce unnecessary computation, fast match algorithms (described in Section 13.2.3) can be used to provide both expected acoustic and language scores for different context-dependent units to result in early pruning of unpromising contexts.

13.2. OTHER EFFICIENT SEARCH TECHNIQUES

Tree structured lexicon represents an efficient framework of manipulation of search space. In this section we present some additional implementation techniques, which can be used to further improve the efficiency of search algorithms. Most of these techniques can be applied to both Viterbi beam search and stack decoding. They are essential ingredients for a practical large-vocabulary continuous speech recognizer.

13.2.1. Using Entire HMM as a State in Search

The state in state-search space based on HMM-trellis computation is, by definition, a Markov state. Phonetic HMM models are the basic unit in most speech recognizers. Even though subphonetic HMMs, like senones, might be used for such a system, the search is often based on phonetic HMMs.

Treating the entire phonetic HMM as a state in state-search has many advantages. The first obvious advantage is that the number of states the search program needs to deal with is smaller. Note that using the entire phonetic HMM does not in effect reduce the number of states in the search. The entire search space is unchanged. All the states within a phonetic HMM are now bundled together. This means that all of them are either kept in the beam, if the phonetic HMM is regarded as promising, or all of them are pruned away. For any given time, the minimum cost among all the states within the phonetic HMM is used as the cost for the phonetic HMM. For pruning purposes, this cost is used to determine the promising

degree of this phonetic HMM, i.e., the fate of all the states within this phonetic HMM. Although this does not actually reduce the beam beyond normal pruning, it has the effect of processing fewer candidates in the beam. In programming, this means less checking and bookkeeping, so some computation savings can be expected.

You might wonder if this organization might be ineffective for beam search, since it forces you to keep or prune all the states within a phonetic HMM. In theory, it is possible that only one or two states in the phonetic HMM need to be kept, while other states can be pruned due to high cost score. However, this is, in reality, very rare, since a phone is a small unit and all the states within a phonetic HMM should be relatively promising when the search is near the acoustic region corresponding to the phone.

During the trellis computation, all the phonetic HMM states need to advance one time step when processing one input vector. By performing HMM computation for all states together, the new organization can reduce memory accesses and improve cache locality, since the output and transition probabilities are held in common by all states. Combining this organization strategy with lexical tree search further enhances the efficiency. In lexical tree search, each hypothesis in the beam is associated with a particular node in the lexical tree. These hypotheses are linked together in the heap structure described in Algorithm 13.2 for the purposes of efficient evaluation and heuristic pruning. Since the node corresponds to a phonetic HMM, the HMM evaluation is guaranteed to execute once for each hypothesis sharing this node.

In summary, treating the entire phonetic HMM as a state in state-search space allows you to explore the effective data structure for better sharing and improved memory locality.

13.2.2. Different Layers of Beams

Because of the complexity of search, it often requires pruning of various levels of search to make search feasible. Most systems thus employ different pruning thresholds to control what states participate. The most frequently used thresholds are listed below:

- τ_s controls what states (either phone states or senone states) to retain. This is the most fundamental beam threshold.
- τ_p controls whether the next phone is extended. Although this might not be necessary for both stack decoding and linear Viterbi beam search, it is crucial for lexical tree search, because pruning unpromising phonetic prefixes in the lexical trees could improve search efficiency significantly.
- τ_h controls whether hypotheses are extended for the next word. Since the branching factor for word boundaries is very large, we need this threshold to limit search to only the promising ones.
- τ_c controls where a linguistic context is created in a lexical tree search using higher-order language models. This is also known as ε-heap in Algorithm 13.2.

Pruning can introduce search errors if a state is pruned that would have been on the globally best path. The principle applied here is that the more constraints you have available, the more aggressively you decide whether this path will participate in the globally best path. In this case, at the state level, you have the least constraints. At the phonetic level there are more, and there are the most at the word level. In general, the number of word hypotheses tends to drop significantly at word boundaries. Different thresholds for different levels allow the search designer to fine-tune those thresholds for their tasks to achieve best search performance without significant increase in error rates.

13.2.3. Fast Match

As described in Chapter 12, fast match is a crucial part of stack decoding, which mainly reduces the number of possible word expansions for each path. Similarly, fast match can be applied to the most expensive part—extending the phone HMM fan-outs within or between lexical trees. Fast match is a method for rapidly deriving a list of candidates that constrain successive search phases in which a computationally expensive *detailed match* is performed. In this sense, fast match can be regarded as an additional pruning threshold to meet before a new word/phone can be started.

Fast match is typically characterized by the approximations that are made in the acoustic/language models to reduce computation. The factorization of language model scores among tree branches in lexical trees described in Section 13.1.3 can be viewed as fast match using a language model. The factorized method is also an admissible estimate of the language model scores for the future word. In this section we focus on acoustic model fast match.

13.2.3.1. Look-Ahead Strategy

Fast match, when applied in time-synchronous search, is also called *look-ahead* strategy, since it basically searches ahead of the time-synchronous search by a few frames to determine which words or phones are likely to extend. Typically the look-ahead frames are fixed, and the fast match is also done in time-synchronous fashion with another specialized beam for efficient pruning. You can also use simplified models, like the one-state HMMs or context-independent models [4, 32]. Some systems [21, 22] have tried to simplify the level of details in the input feature vectors by aggregating information from several frames into one. A straightforward way for compressing the feature stream is to skip every other frame of speech for fast match. This allows a longer-range look-ahead, while keeping computation under control. The approach of simplifying the input feature stream instead of simplifying the acoustic models can reuse the fast match results for detailed match.

Whisper [4] uses phoneme look-ahead fast match in lexical tree search, in which pruning is applied based on the estimation of the score of possible phone fan-outs that may follow a given phone. A context-independent phone-net is searched synchronously with the

search process but offset N frames into the future. In practice, significant savings can be obtained in search efforts without increase in error rates.

The performance of word and phoneme look-ahead clearly depends on the length of the look-ahead frames. In general, the larger the look-ahead window, the longer is the computation and the shorter the word/phone Λ list. Empirically, the window is a few tens of milliseconds for phone look-ahead and a few hundreds of milliseconds for word look-ahead.

13.2.3.2. The Rich-Get-Richer Strategy

For systems employing continuous-density HMMs, tens of mixtures of Gaussians are often used for the output probability distribution for each state. The computation of the mixtures is one of the bottlenecks when many context-dependent models are used. For example, Whisper uses about 120,000 Gaussians. In addition to using various beam pruning thresholds in the search, there could be significant savings if we have a strategy to limit the number of Gaussians to be computed.

The *Rich-Get-Richer* (RGR) strategy enables us to focus on most promising paths and treat them with detailed acoustic evaluations and relaxed path-pruning thresholds. On the contrary, the less promising paths are extended with less expensive acoustic evaluations and less forgiving path-pruning thresholds. In this way, locally optimal candidates continue to receive the maximum attention while less optimal candidates are retained but evaluated using less precise (computationally expensive) acoustic and/or linguistic models. The RGR strategy gives us finer control in the creation of new paths that has potential to grow exponentially.

RGR is used to control the level of acoustic details in the search. The goal is to reduce the number of context-dependent senone probability (Gaussian) computations required. The context-dependent senones associated with a phone instance p would be evaluated according to the following condition:

$$\begin{aligned} &Min[ci(p)] * \alpha + LookAhead[ci(p)] < \text{threshold} \\ &\text{where } Min[ci(p)] = \min_{s}\{\cos t(s) \mid s \in ci\_phone(p)\} \\ &\text{and } \quad LookAhead[ci(p)] = \text{look-ahead estimate of } ci(p) \end{aligned} \tag{13.6}$$

These conditions state that the context-dependent senones associated with p should be evaluated if there exists a state s corresponding to p, whose cost in linear combination with a look-ahead cost score corresponding to p falls within a threshold. In the event that p does not fall within the threshold, the senone scores corresponding to p are estimated using the context-independent senones corresponding to p. This means the context-dependent senones are evaluated only if the corresponding context-independent senones and the look-ahead start showing promise. RGR strategy should save significant senone computation for clearly unpromising paths. Whisper [26] reports that 80% of senone computation can be avoided without introducing significant errors for a 20,000-word WSJ dictation task.

13.3. *N*-BEST AND MULTIPASS SEARCH STRATEGIES

Ideally, a search algorithm should consider all possible hypotheses based on a unified probabilistic framework that integrates all *knowledge sources* (KSs).[5] These KSs, such as acoustic models, language models, and lexical pronunciation models, can be integrated in an HMM state search framework. It is desirable to use the most detailed models, such as context-dependent models, interword context-dependent models, and high-order n-grams, in the search as early as possible. When the explored search space becomes unmanageable, due to the increasing size of vocabulary or highly sophisticated KSs, search might be infeasible to implement.

As we develop more powerful techniques, the complexity of models tends to increase dramatically. For example, language understanding models in Chapter 17 require long-distance relationships. In addition, many of these techniques are not operating in a left-to-right manner. A possible alternative is to perform a multipass search and apply several KSs at different stages, in the proper order to constrain the search progressively. In the initial pass, the most discriminant and computationally affordable KSs are used to reduce the number of hypotheses. In subsequent passes, progressively reduced sets of hypotheses are examined, and more powerful and expensive KSs are then used until the optimal solution is found.

The early passes of multipass search can be considered fast match that eliminates those unlikely hypotheses. Multipass search is, in general, not admissible because the optimal word sequence could be wrongly pruned prematurely, due to the fact that not all KSs are used in the earlier passes. However, for complicated tasks, the benefits of computation complexity reduction usually outweigh the nonadmissibility. In practice, multipass search strategy using progressive KSs could generate better results than a search algorithm forced to use less powerful models due to computation and memory constraints.

The most straightforward multipass search strategy is the so-called n-best search paradigm. The idea is to use affordable KSs to first produce a list of n most probable word sequences in a reasonable time. Then these n hypotheses are rescored using more detailed models to obtain the most likely word sequence. The idea of the n-best list can be further extended to create a more compact hypotheses representation—namely word lattice or graph. A word lattice is a more efficient way to represent alternative hypotheses. *N*-best or lattice search is used for many large-vocabulary continuous speech recognition systems [20, 30, 44].

In this section we describe the representation of the n-best list and word lattice. Several algorithms to generate such an n-best-list or word lattice are discussed.

[5] In the field of artificial intelligence, the process of performing search through an integrated network of various knowledge sources is called *constraint satisfaction*.

13.3.1. *N*-best Lists and Word Lattices

Table 13.4 shows an example n-best (10-best) list generated for a North American Business (NAB) sentence. *N*-best search framework is effective only for n of the order of tens or hundreds. If the short n-best list that is generated by using less optimal models does not include the correct word sequence, the successive rescoring phases have no chance to generate the correct answer. Moreover, in a typical n-best list like the one shown in Table 13.4, many of the different word sequences are just one-word variations of each other. This is not surprising, since similar word sequences should achieve similar scores. In general, the number of n-best hypotheses might grow exponentially with the length of the utterance. Word lattices and word graphs are thus introduced to replace n-best list with a more compact representation of alternative hypotheses.

Word lattices are composed by word hypotheses. Each word hypothesis is associated with a score and an explicit time interval. Figure 13.8 shows an example of a word lattice corresponding to the n-best list example in Table 13.4. It is clear that a word lattice is more efficient representation. For example, suppose the spoken utterance contains 10 words and there are 2 different word hypotheses for each word position. The n-best list would need to have $2^{10} = 1024$ different sentences to include all the possible permutations, whereas the word lattice requires only 20 different word hypotheses.

Word graphs, on the other hand, resemble finite state automata, in which arcs are labeled with words. Temporal constraints between words are implicitly embedded in the topology. Figure 13.9 shows a word graph corresponding to the n-best list example in Table 13.4. Word graphs in general have an explicit specification of word connections that don't allow overlaps or gaps along the time axis. Nonetheless, word lattices and graphs are similar, and we often use these terms interchangeably.[6] Since an n-best list can be treated as a simple word lattice, word lattices are a more general representation of alternative hypotheses. *N*-best lists or word lattices are generally evaluated on the following two parameters:

Table 13.4 An example 10-best list for a North American Business sentence.

1.	I will tell you would I think in my office
2.	I will tell you what I think in my office
3.	I will tell you when I think in my office
4.	I would sell you would I think in my office
5.	I would sell you what I think in my office
6.	I would sell you when I think in my office
7.	I will tell you would I think in my office
8.	I will tell you why I think in my office
9.	I will tell you what I think on my office
10.	I Wilson you I think on my office

[6] We will use the term *word lattice* in the rest of this chapter..

- *Density:* In the n-best case, it is measured by how many alternative word sequences are kept in the n-best list. In the word lattice case, it is measured by the number of word hypotheses or word arcs per uttered word. Obviously, we want the density to be as small as possible for successive rescoring modules, provided the correct word sequence is included in the n-best list or word lattice.
- *The lower bound word error rate:* It is the lowest word error rate for any word sequence in the n-best list or the word lattice.

I	will	tell	you	what	I	think	in	my	office
	would	sell		when					
	Wilson			why					
				would					
				I					

Figure 13.8 A word lattice example. Each word has an explicit time interval associated with it.

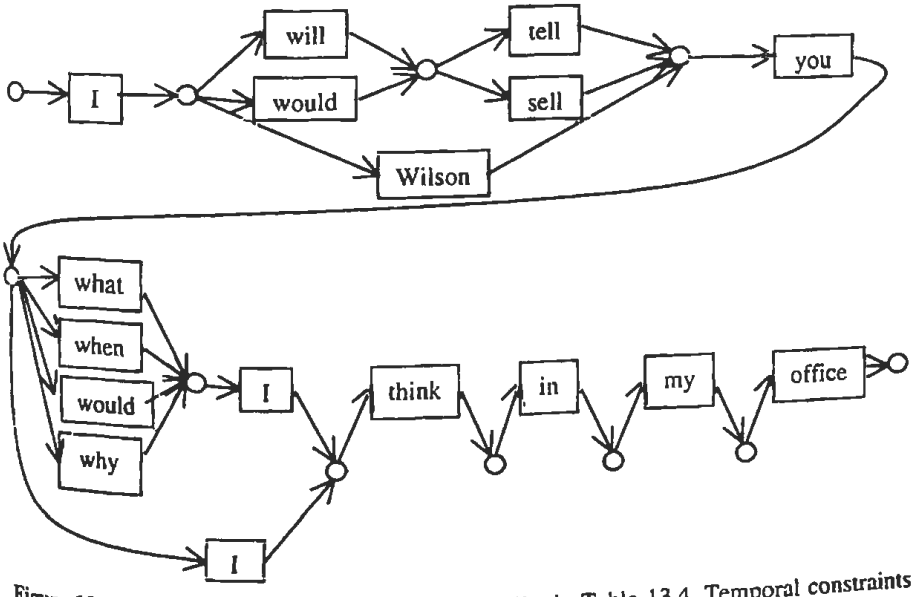


Figure 13.9 A word graph example for the n-best list in Table 13.4. Temporal constraints are implicit in the topology.

Rescoring with highly similar n-best alternatives duplicates computation on common parts. The compact representation of word lattices allows both data structure and computation sharing of the common parts among similar alternative hypotheses, so it is generally computationally less expensive to rescore the word lattice.

Figure 13.10 illustrates the general n-best/lattice search framework. Those KSs providing most constraints, at a lesser cost, are used first to generate the n-best list or word lattice. The n-best list or word lattice is then passed to the rescoring module, which uses the remaining KSs to select the optimal path. You should note that the n-best and word-lattice generators sometimes involve several phases of search mechanisms to generate the n-best list or word lattice. Therefore, the whole search framework in Figure 13.10 could involve several (> 2) phases of search mechanism.

Does the compact n-best or word-lattice representation impose constraints on the complexity of the acoustic and language models applied during successive rescoring modules? The word lattice can be expanded for higher-order language models and detailed context-dependent models, like inter-word triphone models. For example, to use higher-order language models for word lattice entails copying each word in the appropriate context of preceding words (in the trigram case, the two immediately preceding words). To use interword triphone models entails replacing the triphones for the beginning and ending phone of each word with appropriate interword triphones. The expanded lattice can then be used with detailed acoustic and language models. For example, Murveit et al. [30] report this can achieve trigram search without exploring the enormous trigram search space.

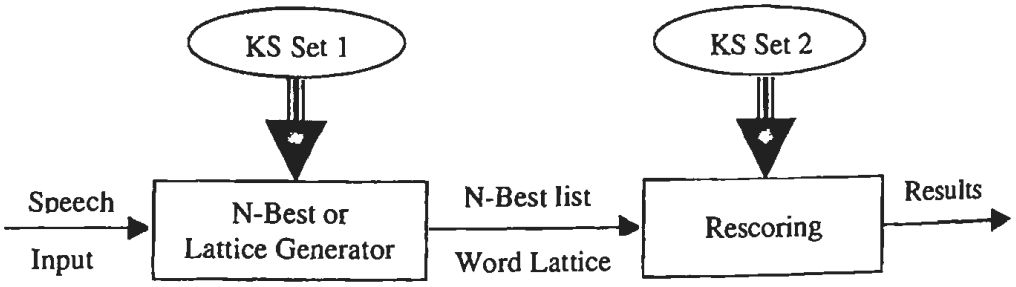


Figure 13.10 N-best/lattice search framework. The most discriminant and inexpensive knowledge sources (KSs 1) are used first to generate the n-best/lattice. The remaining knowledge sources (KSs 2, usually expensive to apply) are used in the rescoring phase to pick up the optimal solution [40].

13.3.2. The Exact N-best Algorithm

Stack decoding is the choice of generating n-best candidates because of its best-first principle. We can keep it generating results until it finds n complete paths; these n complete sentences form the n-best list. However, this algorithm usually cannot generate the n best candidates efficiently. The efficient n-best algorithm for time-synchronous Viterbi search was first introduced by Schwartz and Chow [39]. It is a simple extension of time-synchronous Viterbi search. The fundamental idea is to maintain separate records for paths

with distinct histories. The history is defined as the whole word sequence up to the current time t and word w. This exact n-best algorithm is also called *sentence-dependent* n-best algorithm. When two or more paths come to the same state at the same time, paths having the same history are merged and their probabilities are summed together; otherwise, only the n-best paths are retained for each state. As commonly used in speech recognition, a typical HMM state has 2 or 3 predecessor states within the word HMM. Thus, for each time frame and each state, the n-best search algorithm needs to compare and merge 2 or 3 sets of n paths into n new paths. At the end of the search, the n paths in the final state of the trellis are simply re-ordered to obtain the n-best word sequences.

This straightforward n-best algorithm can be proved to be admissible[7] in normal circumstances [40]. The complexity of the algorithm is proportional to $O(n)$, where n is the number of paths kept at each state. This is often too slow for practical systems.

13.3.3. Word-Dependent *N*-best and Word-Lattice Algorithm

Since many of the different entries in the n-best list are just one-word variations of each other, as shown in Table 13.4, one efficient algorithm can be derived from the normal 1-best Viterbi algorithm to generate the n-best hypotheses. The algorithm runs just like the normal time-synchronous Viterbi algorithm for all within-word transitions. However for each time frame t, and each word-ending state, the algorithm stores all the different words that can end at current time t and their corresponding scores in a *traceback* list. At the same time, the score of the best hypothesis at each grammar state is passed forward, as in the normal time-synchronous Viterbi search. This obviously requires almost no extra computation above the normal time-synchronous Viterbi search. At the end of search, you can simply search through the stored traceback list to get all the permutations of word sequences with their corresponding scores. If you use a simple threshold, the traceback can be implemented very efficiently to only uncover the word sequences with accumulated cost scores below the threshold. This algorithm is often referred as *traceback*-based n-best algorithm [29, 42] because of the use of the traceback list in the algorithm.

However, there is a serious problem associated with this algorithm. It could easily miss some low-cost hypotheses. Figure 13.11 illustrates an example in which word w_k can be preceded by two different words w_i and w_j in different time frames. Assuming path w_i-w_k has a lower cost than path w_j-w_k when both paths meet during the trellis search of w_k, the path w_j-w_k will be pruned away. During traceback for finding the n-best word sequences, there is only one best starting time for word w_k, determined by the best boundary between the best preceding word w_i and it. Even though path w_j-w_k might have a very low cost (let's say only marginally higher than that of w_i-w_k), it could be completely overlooked, since the path has a different starting time for word w_k.

[7] Although one can show, in the worst case, when paths with different histories have near identical scores for each state, the search actually needs to keep all paths ($> N$) in order to guarantee absolute admissibility. Under this worst case, the admissible algorithm is clearly exponential in the number of words for the utterance, since all permutations of word sequences for the whole sentence need to be kept.

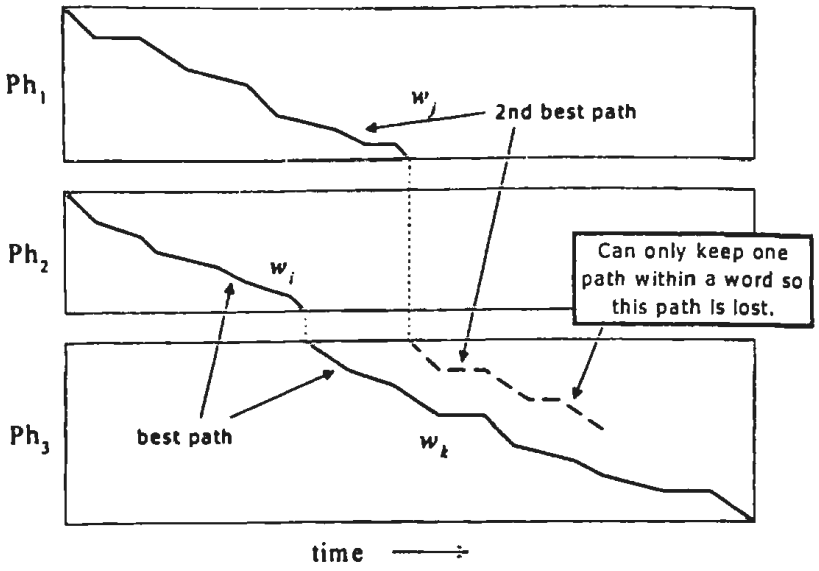


Figure 13.11 Deficiency in traceback-based n-best algorithm. The best subpath, $w_i - w_k$, will prune away subpath $w_j - w_k$ while searching the word w_k; the second-best subpath cannot be recovered [40].

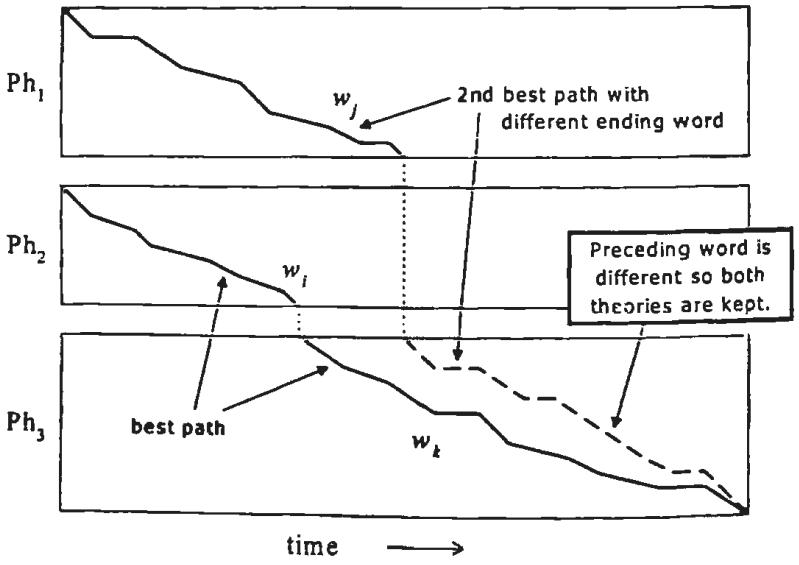


Figure 13.12 Word-dependent n-best algorithm. Both subpaths $w_i - w_k$ and $w_j - w_k$ are kept under the word-dependent assumption [40].

The *word-dependent* n-best algorithm [38] can alleviate the deficiency of the traceback-based n-best algorithm, in which only one starting time is kept for each word, so the starting time is independent of the preceding words. On the other hand, in the sentence-dependent n-best algorithm, the starting time for a word depends on all the preceding words, since different histories are kept separately. A good compromise is the so-called word-dependent assumption: *The starting time of a word depends only on the immediate preceding word.* That is, given a word pair and its ending time, the boundary between these two words is independent of further predecessor words.

In the word-dependent assumption, the history to be considered for a different path is no longer the entire word sequence; instead, it is only the immediately preceding word. This allows you to keep k ($<< n$) different records for each state and each time frame in Viterbi search. Differing slightly from the exact n-best algorithm, a traceback must be performed to find the n-best list at the end of search. The algorithm is illustrated in Figure 13.12. A word-dependent n-best algorithm has a time complexity proportional to k. However, it is no longer admissible because of the word-dependent approximation. In general, this approximation is quite reasonable if the preceding word is long. The loss it entails is insignificant [6].

13.3.3.1. One-Pass *N*-best and Word-Lattice Algorithm

As presented in Section 13.1, one-pass Viterbi beam search can be implemented very efficiently using a tree lexicon. Section 13.1.2 states that multiple copies of lexical trees are necessary for incorporating language models other than the unigram. When bigram is used in lexical tree search, the successor lexical tree is predecessor-dependent. This predecessor-dependent property immediately translates into the word-dependent property,[8] as defined in Section 13.3.3, because the starting time of a word clearly depends on the immediately preceding word. This means that different word-dependent partial paths are automatically saved under the framework of predecessor-dependent successor trees. Therefore, one-pass predecessor-dependent lexical tree search can be modified slightly to output n-best lists or word graphs.

Ney et al. [31] used a word graph builder with a one-pass predecessor-dependent lexical tree search. The idea is to exploit the word-dependent property inherited from the predecessor-dependent lexical tree search. During predecessor-dependent lexical tree search, two additional quantities are saved whenever a word ending state is processed.

$\tau(t;w_i,w_j)$—Representing the optimal word boundary between word w_i and w_j, given word w_j ending at time t.

$h(w_j;\tau(t;w_i,w_j),t) = -\log P(\mathbf{x}_\tau^t \mid w_j)$—Representing the cumulative cost that word w_j produces acoustic vector $\mathbf{x}_\tau, \mathbf{x}_{\tau+1}, \cdots \mathbf{x}_t$.

[8] When higher order n-gram models are used, the boundary dependence will be even more significant. For example, when trigrams are used, the boundary for a word juncture depends on the previous two words. Since we generally want a fast method of generating word lattices/graphs, bigram is often used instead of higher order n-gram to generate word lattices/graphs.

At the end of the utterance, the word lattice or n-best list is constructed by tracing back all the permutations of word pairs recorded during the search. The algorithm is summarized in Algorithm 13.3.

ALGORITHM 13.3: *ONE-PASS PREDECESSOR-DEPENDENT LEXICAL TREE SEARCH FOR N-BEST OR WORD-LATTICE CONSTRUCTION*

Step 1: For $t = 1..T$,
    1-best predecessor-dependent lexical tree search;
    $\forall(w_i, w_j)$ ending at t
        record word-dependent crossing time $\tau(t; w_i, w_j)$;
        record cumulative word score $h(w_j; \tau(t; w_i, w_j), t)$;
Step 2: Output 1-best result;
Step 3: Construct n-best or word-lattice by tracing back the word-pair records (τ and h).

13.3.4. The Forward-Backward Search Algorithm

As described Chapter 12, the ability to predict how well the search fares in the future for the remaining portion of the speech helps to reduce the search effort significantly. The one-pass search strategy, in general, has very little chance of predicting the cost for the portion that it has not seen. This difficulty can be alleviated by multipass search strategies. In successive phases the search should be able to provide good estimates for the remaining paths, since the entire utterance has been examined by the earlier passes. In this section we investigate a special type of multipass search strategy—forward-backward search.

The idea is to first perform a forward search, during which partial forward scores α for each state can be stored. Then perform a second pass search backward—that is, the second pass starts by taking the final frame of speech and searches its way back until it reaches the start of the speech. During the backward search, the partial forward scores α can be used as an accurate estimate of the heuristic function or the fast match score for the remaining path. Even though different KSs might be used in forward and backward phases, this estimate is usually close to perfect, so the search effort for the backward phase can be significantly reduced.

The forward search must be very fast and is generally a time-synchronous Viterbi search. As in the multipass search strategy, simplified acoustic and language models are often used in forward search. For backward search, either time-synchronous search or time-asynchronous A* search can be employed to find the n-best word sequences or word lattice.

13.3.4.1. Forward-Backward Search

Stack decoding, as described in Chapter 12, is based on the admissible A* search, so the first complete hypothesis found with a cost below that of all the hypotheses in the stack is guaranteed to be the best word sequence. It is straightforward to extend stack decoding to produce the n-best hypotheses by continuing to extend the partial hypotheses according to the same A* criterion until n different hypotheses are found. These n different hypotheses are destined to be the n-best hypotheses under a proof similar to that presented in Chapter 12. Therefore, stack decoding is a natural choice for producing the n-best hypotheses.

However, as described in Chapter 12, the difficulty of finding a good heuristic function that can accurately under-estimate the remaining path has limited the use of stack decoding. Fortunately, this difficulty can be alleviated by *tree-trellis forward-backward search* algorithms [41]. First, the search performs a time-synchronous forward search. At each time frame t, it records the score of the final state of each word ending. The set of words whose final states are active (surviving in the beam) at time t is denoted as Δ_t. The score of the final state of each word w in Δ_t is denoted as $\alpha_t(w)$, which represents the sum of the cost of matching the utterance up to time t given the most likely word sequence ending with word w and the cost of the language model score for that word sequence. At the end of the forward search, the best cost is obtained and denoted as α^T.

After the forward pass is completed, the second search is run in reverse (backward), i.e., considering the last frame T as the beginning one and the first frame as the final one. Both the acoustic models and language models need to be reversed. The backward search is based on A* search. At each time frame t, the best path is removed from the stack and a list of possible one-word extensions for that path is generated. Suppose this best path at time t is ph_{w_j}, where w_j is the first word of this partial path (the last expanded during backward A* search). The exit score of path ph_{w_j} at time t, which now corresponds to the score of the initial state of the word HMM w_j, is denoted as $\beta_t(ph_{w_j})$.

Let us now assume we are concerned about the one-word extension of word w_i for path ph_{w_j}. Remember that there are two fundamental issues for the implementation of A* search algorithm—(1) finding an effective and efficient heuristic function for estimating the future remaining input feature stream and (2) finding the best crossing time between w_i and w_j.

The stored forward score α can be used for solving both issues effectively and efficiently. For each time t, the sum $\alpha_t(w_i)+\beta_t(ph_{w_j})$ represents the cost score of the best complete path including word w_i and partial path ph_{w_j}. $\alpha_t(w_i)$ clearly represents a very good heuristic estimate of the remaining path from the start of the utterance until the end of the word w_i, because it is indeed the best score computed in the forward path for the same quantity. Moreover, the optimal crossing time t^* between w_i and w_j can be easily computed by the following equation:

$$t^* = \arg\min_t \left[\alpha_t(w_i)+\beta_t(ph_{w_j})\right] \tag{13.7}$$

Finally, the new path ph', including the one-word (w_i) extension, is inserted into the stack, ordered by the cost score $\alpha_t(w_i)+\beta_t(ph_{w_i})$. The heuristic function (forward scores α) allows the backward A* search to concentrate search on extending only a few truly promising paths.

As a matter of fact, if the same acoustic and language models are used in both the forward and backward search, this heuristic estimate (forward scores α) is indeed a perfect estimate of the best score the extended path will achieve. The first complete hypothesis generated by backward A* search coincides with the best one found in the time-synchronous forward search and is truly the best hypothesis. Subsequent complete hypotheses correspond sequentially to the n-best list, as they are generated in increasing order of cost. Under this condition, the size of the stack in the backward A* search need only be N. Since the estimate of future is exact, the $(N+1)^{\text{th}}$ path in the stack has no chance to become part of the n-best list. Therefore, the backward search is executed very efficiently to obtain the n-best hypotheses without exploring many unpromising branches. Of course, tree-trellis forward-backward search can also be used like most other multipass search strategies—inexpensive KSs are used in the forward search to get an estimate of α, and more expensive KSs are used in the backward A* search to generate the n-best list.

The same idea of using forward score α can be applied to time-synchronous Viterbi search in the backward search instead of backward A* search [7, 34]. For large-vocabulary tasks, the backward search can run 2 to 3 orders of magnitude faster than a normal Viterbi beam search. To obtain the n-best list from time-synchronous forward-backward search, the backward search can also be implemented in a similar way as a time-synchronous word-dependent n-best search.

13.3.4.2. Word-Lattice Generation

The forward-backward n-best search algorithm can be easily modified to generate word lattices instead of n-best lists. A forward time-synchronous Viterbi search is performed first to compute $\alpha_t(\omega)$, the score of each word ω ending at time t. At the end of the search, this best score α^T is also recorded to establish the global pruning threshold. Then, a backward time-synchronous Viterbi search is performed to compute $\beta_t(\omega)$, the score of each word ω beginning at time t. To decide whether to include word juncture $\omega_i-\omega_j$ in the word lattice/graph at time t, we can check whether the forward-backward score is below a global pruning threshold. Specifically, supposed bigram probability $P(\omega_j \mid \omega_i)$ is used, if

$$\alpha_t(\omega_i)+\beta_t(\omega_j)+\left[-\log P(\omega_j \mid \omega_i)\right] < \alpha^T+\theta \tag{13.8}$$

where θ is the pruning threshold, we will include $\omega_i-\omega_j$ in the word lattice/graph at time t. Once word juncture $\omega_i-\omega_j$ is kept, the search continues looking for the next word-pair, where the first word ω_i will be the second word of the next word-pair.

The above formulation is based on the assumption of using the same acoustic and language models in both forward and backward search. If different KSs are used in forward and backward search, the normalized α and β scores should be used instead.

13.3.5. One-Pass vs. Multipass Search

There are several real-time one-pass search engines [4, 5]. Is it necessary to build a multipass search engine based on n-best or word-lattice rescoring? We address this issue by discussing the disadvantages and advantages of multipass search strategies.

One criticism of multipass search strategies is that they are not suitable for real-time applications. No matter how fast the first pass is, the successive (backward) passes cannot start until users finish speaking. Thus, the search results need to be delayed for at least the time required to execute the successive (backward) passes. This is why the successive passes must be extremely fast in order to shorten the delay. Fortunately, it is possible to keep the delays minimum (under one second) with clever implementation of multipass search algorithms, as demonstrated by Nguyen et al. [18].

Another criticism for multipass search strategies is that each pass has the potential to introduce inadmissible pruning, because decisions made in earlier passes are based on simplified models (KSs). Search is a constraint-satisfaction problem. When a pruning decision in each search pass is made on a subset of constraints (KSs), pruning error is inevitable and is unrecoverable by successive passes. However, inadmissible pruning, like beam pruning and fast match, is often necessary to implement one-pass search in order to cope with the large active search space caused jointly by complex KSs and large-vocabulary tasks. Thus, the problem of inadmissibility is actually shared by both real-time one-pass search and multipass search for different reasons. Fortunately, in both cases, search errors can be reduced to a minimum by clever implementation and by empirically designing all the pruning thresholds carefully, as demonstrated in various one-pass and multipass systems [4, 5, 18].

Despite these concerns regarding multipass search strategies, they remain important components in developing spoken language systems. We list here several important aspects:

1. It might be necessary to use multipass search strategies to incorporate very expensive KSs. Higher-order n-gram, long-distance context-dependent models, and natural language parsing are examples that make the search space unmanageable for one-pass search. Multipass search strategies might be compelling even for some small-vocabulary tasks. For example, there are only a couple of million legal credit card numbers for the authentication task of 16-digit credit card numbers. However, it is very expensive to incorporate all the legal numbers explicitly in the recognition grammar. To first reduce search space down to an n-best list or word lattice/graph might be a desirable approach.
2. Multipass search strategies could be very compelling for spoken language understanding systems. It is problematic to incorporate most natural language

understanding technologies in one-pass search. On the other hand, n-best lists or word lattices provide a trivial interface between speech recognition and natural language understanding modules. Such an interface also provides a convenient mechanism for integrating different KSs in a modular way. This is important because the KSs could come from different modalities (like video or pen) that make one-pass integration almost infeasible. This high degree of modality allows different component subsystems to be optimized and implemented independently.

3. N-best lists or word lattices are very powerful offline tools for developing new algorithms for spoken language systems. It is often a significant task to fully integrate new modeling techniques, such as segment models, into a one-pass search. The complexity could sometimes slow down the progress of the development of such techniques, since recognition experiments are difficult to conduct. Rescoring of n-best list and lattice provides a quick and convenient alternative for running recognition experiments. Moreover, the computation and storage complexity can be kept relatively constant for offline n-best or word lattice/graph search strategies even when experimenting with highly expensive new modeling techniques. New modeling techniques can be experimented with using n-best/word-graph framework first, being integrated into the system only after significant improvement is demonstrated.
4. Besides being an alternative search strategy, n-best generation is also essential for discriminant training. Discriminant training techniques, like MMIE, and MCE described in Chapter 4, often need to compute statistics of all possible rival hypotheses. For isolated word recognition using word models, it is easy to enumerate all the word models as the rival hypotheses. However, for continuous speech recognition, one needs to use an all-phone or all-word model to generate all possible phone sequences or all possible word sequences during training. Obviously, that is too expensive. Instead, one can use n-best search to find all the near-miss sentence hypotheses that we want to discriminate against [15, 36].

13.4. SEARCH-ALGORITHM EVALUATION

Throughout this chapter we are careful in following dynamic programming principles, using admissible criteria as much as possible. However, many heuristics are still unavoidable to implement large-vocabulary continuous speech recognition in practice. Those nonadmissible heuristics include:

- Viterbi score instead of forward score described in Chapter 12.
- Beam pruning or stack pruning described in Section 13.2.2 and Chapter 12.

- Subtree dominance pruning described in Section 13.1.5.
- Fast match pruning described in Section 13.2.3.
- Rich-get-richer pruning described in Section 13.2.3.2.
- Multipass search strategies described in Section 13.3.5.

Nonadmissible heuristics generate suboptimal searches where the found path is not necessarily the path with the minimum cost. The question is, how different is this suboptimal from the true optimal path? Unfortunately, there is no way to know the optimal path unless an exhaustive search is conducted. The practical question is whether the suboptimal search hurts the search result. In a test condition where the true result is specified, you can easily compare the search result with the true result to find whether any error occurs. Errors could be due to inaccurate models (including acoustic and language models), suboptimal search, or end-point detection. The error caused by a suboptimal search algorithm is referred to as *search error* or *pruning error*.

How can we find out whether the search commits a pruning error? One of the procedures most often used is straightforward. Let $\hat{W}$ be the recognized word sequence from the recognizer and $\tilde{W}$ be the true word sequence. We need to compare the cost for these two word sequences:

$$-\log P(\hat{W}|\mathbf{X}) \propto -\log\left[P(\hat{W})P(\mathbf{X}|\hat{W})\right] \tag{13.9}$$

$$-\log P(\tilde{W}|\mathbf{X}) \propto -\log\left[P(\tilde{W})P(\mathbf{X}|\tilde{W})\right] \tag{13.10}$$

The quantity in Eq. (13.9) is supposed to be minimum among all possible word sequences if the search is admissible. Thus, if the quantity in Eq. (13.10) is greater than that in Eq. (13.9), the error is not attributed to search pruning. On the other hand, if the quantity in Eq. (13.10) is smaller than that in Eq. (13.9), there is a search error. The rationale behind the procedure described here is obvious. In the case of search errors, the suboptimal search (or nonadmissible pruning) has obviously pruned the correct path, because the cost of the correct path is smaller than the one found by the recognizer. Although we can conclude that search errors are found in this case, it does not guarantee that the search result is correct if the search can be made optimal. The reason is simply that there might be one pruned path with an incorrect word sequence and lower cost under the same suboptimal search. Therefore, the search errors represent only the upper bound that one can improve on if an optimal search is carried out. Nonetheless, finding search errors by comparing quantities in Eqs. (13.9) and (13.10) is a good measure in different search algorithms.

During the development of a speech recognizer, it is a good idea to always include the correct path in the search space. By including such a path, and some bookkeeping, one can use the correct path to help in determining all the pruning thresholds. If the correct path is pruned away during search by some threshold, some adjustment can be made to relax such a

threshold to retain the correct path. For example, one can adjust the pruning threshold for fast match if a word in $\tilde{W}$ fails to appear on the list supplied by the fast match.

13.5. CASE STUDY—MICROSOFT WHISPER

We use the decoder of Microsoft's Whisper [26, 27] discussed in Chapter 9 as a case study for reviewing the search techniques we have presented in this chapter. Whisper can handle both context-free grammars for small-vocabulary tasks and n-gram language models for large-vocabulary tasks. We describe these two different cases.

13.5.1. The CFG Search Architecture

Although context-free grammars (CFGs) have the disadvantage of being too restrictive and unforgiving, particularly with novice users, they are still one of the most popular configurations for building limited-domain applications because of the following advantages:

- Compact representation results in a small memory footprint.
- Efficient operation during decoding in terms of both space and time.
- Ease of grammar creation and modification for new tasks.

As mentioned in Chapter 12, the CFG grammar consists of a set of productions or rules that expand nonterminals into a sequence of terminals and nonterminals. Nonterminals in the grammar tend to refer to high-level task-specific concepts such as dates, font names, and commands. The terminals are words in the vocabulary. A grammar also has a nonterminal designated as its start state. Whisper also allows some regular expression operators on the right-hand side of the production for notational convenience. These operators are: or '|'; repeat zero or more times '*'; repeat one or more times '+'; and optional ([ ]). The following is a simple CFG example for *binary number:*

```
%start BINARY_NUMBER
BINARY_NUMBER: (zero | one)*
```

Without losing generality, Whisper disallows the left recursion for ease of implementation [2]. The grammar is compiled into a binary linked list format. The binary format currently has a direct one-to-one correspondence with the text grammar components, but is more compact. The compiled format is used by the search engine during decoding. The binary representation consists of variable-sized nodes linked together. The grammar format achieves sharing of subgrammars through the use of shared nonterminal definition rules.

The CFG search is conducted according to RTN framework (see Chapter 12). During decoding, the search engine pursues several paths through the CFG at the same time. Associated with each of the paths is a grammar state that describes completely how the path can be extended further. When the decoder hypothesizes the end of the current word of a path, it asks the grammar module to extend the path further by one word. There may be several alternative successor words for the given path. The decoder considers all the successor word

possibilities. This may cause the path to be extended to generate several more paths to be considered, each with its own grammar state. Also note that the same word might be under consideration by the decoder in the context of different paths and grammar states at the same time.

The decoder uses beam search to prune unpromising paths with three different beam thresholds. The state pruning threshold τ_s and new phone pruning threshold τ_p work as described in Section 13.2.2. When extending a path, if the score of the extended path does not exceed the threshold τ_h, the path to be extended is put into a pool. At each frame, for each word in the vocabulary, a winning path that extends to that word is picked from the pool, based on the score. All the remaining paths in the pool are pruned. This level of pruning gives us finer control in the creation of new paths that have potential to grow exponentially.

When two paths representing different word sequences thus far reach the end of the current word with the same grammar state at the same time, only the better path of the two is allowed to continue on. This optimization is safe, except that it does not take into account the effect of different interword left acoustic contexts on the scores of the new word that is started.

Besides beam pruning, the RGR strategy, described in Section 13.2.3.2, is used to avoid unnecessary senone computation. The basic idea is to use the linear combination of context-independent senone score and context-independent look-ahead score to determine whether the context-dependent senone evaluation is worthwhile to pursue.

All of these pruning techniques enable Whisper to perform typical 100- to 200-word CFG tasks in real time running on a 486 PC with 2 MB RAM. Readers might think it is not critical to make CFG search efficient on such a low-end platform.[9] However, it is indeed important to keep the CFG engine fast and lean. The speech recognition engine is eventually only part of an integrated application. The application will benefit if the resources (both CPU and memory) used by the speech decoder are kept as small as possible, so there are more resources left for the application module to use. Moreover, in recognition server applications, several channels of speech recognition can be performed on a single server platform if each speech recognition engine takes only a small portion of the total resources.

13.5.2. The *N*-gram Search Architecture

The CFG decoder is best suited for limited domain command and control applications. For dictation or natural conversational systems, a stochastic grammar such as n-grams provides a more natural choice. Using bigrams or trigrams leads to a large number of states to be considered by the search process, requiring an alternative search architecture.

[9] Thanks to the progress predicted by Moore's law, the current mainstream PC configuration is an order of magnitude more powerful than the configuration we list here (486 PC with 2 MB RAM) in both speed and memory.

Whisper's n-gram search architecture is based on lexical tree search as described in Section 13.1. To keep the runtime memory[10] as small as possible, Whisper does not need to allocate the entire lexical tree network statically. Instead, it dynamically builds only the portion that needs to be active. To cope with the problem of delayed application of language model scores, Whisper uses the factorization algorithm described in Section 13.1.3 to distribute the language model probabilities through the tree branches. To reduce the memory overhead of the factored language model probabilities, an efficient data structure is used for representing the lexical tree as described in Section 13.1.3.1. This data structure allows Whisper to encode factored language model probabilities in no more than the space required for the original n-gram probabilities. Thus, there is absolutely no storage overhead for using factored lexical trees.

The basic acoustic subword model in Whisper is a context-dependent senone. It also incorporates inter-word triphone models in the lexical tree search as described in Section 13.1.6. Table 13.5 shows the distribution of phoneme arcs for 20,000-word WSJ lexical tree using senones as acoustic models. Context-dependent units certainly prohibit more prefix sharing when compared with Table 13.1. However, the overall arcs in the lexical tree still represent quite a saving when compared with a linear lexicon with about 140,000 phoneme arcs. Most importantly, similar to the case in Table 13.1, most sharing is realized in the beginning prefixes where most computation resides. Moreover, with the help of context-dependent and interword senone models, the search is able to use more reliable knowledge to perform efficient pruning. Therefore, lexical tree with context-dependent models can still enjoy all the benefits associated with lexical tree search.

The search organization is evaluated on the 1992 development test set for the *Wall Street Journal* corpus with a back-off trigram language model. The trigram language model has on the order of 10^7 linguistic equivalent classes, but the number of classes generated is far fewer due to the constraints provided by the acoustic model. Figure 13.13(a) illustrates that the relative effort devoted to the trigram, bigram, and unigram is constant regardless of total search effort, across a set of test utterances. This is because the ratio of states in the language model is constant. The language model is using $\sim 2 \times 10^6$ trigrams, $\sim 2 \times 10^6$ bigrams, and 6×10^4 unigrams. Figure 13.13(b) illustrates different relative order when word hypotheses are considered. The most common context for word hypotheses is the unigram context, followed by the bigram and trigram contexts. The reason for the reversal from the state-level transitions is the partially overlapping evaluations required by each bigram context. The trigram context is more common than the bigram context for utterances that generate few hypotheses overall. This is likely because the language model models those utterances well.

[10] Here the runtime memory means the virtual memory for the decoder that is the entire image of the decoder.

Table 13.5 Configuration of the first seven levels of the 20,000-word WSJ (*Wall Street Journal*) tree; the large initial fan-out is due to the use of context-dependent acoustic models [4].

Tree Level	Number of Nodes	Fan-Out
1	655	655.0
2	3174	4.85
3	9388	2.96
4	13,703	1.46
5	14,918	1.09
6	13,907	0.93
7	11,389	0.82

To improve efficiency in dealing with tree copies due to the use of higher-order *n*-gram, one needs to reduce redundant computations in subtrees that are not explicitly part of the given linguistic context. One solution is to use successor trees to include only nonzero successors, as described in Section 13.1.2. Since Whisper builds the search space dynamically, it is not effective for Whisper to use the optimization techniques of the successor-tree network, such as FSN optimization, subtree isomorphism, and sharing tail optimization. Instead, Whisper uses polymorphic linguistic context assignment to reduce redundancy, as described in Section 13.1.5. This involves keeping a single copy of the lexical tree, so that each node in the tree is evaluated at most once. To avoid early inadmissible pruning of different linguistic contexts, an ε-heap of storing paths of different linguistic contexts is created for each node in the tree. The operation of such ε-heaps is in accordance with Algorithm 13.2. The depth of each heap varies dynamically according to a changing threshold that allows more contexts to be retained for promising nodes.

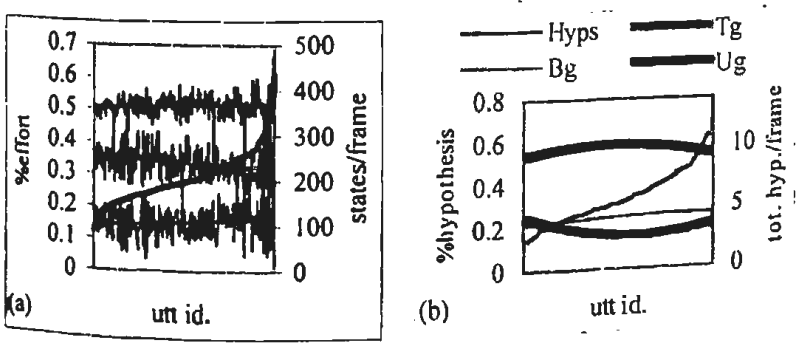


Figure 13.13 (a) Search effort for different linguistic contexts measured by number of active states in each of the three different linguistic contexts. The top series is for the bigram, then the unigram and trigram. The remaining series is the effort per utterance and is plotted on the secondary *y*-axis. (b) The distribution of word hypotheses with respect to their context. The top line is the unigram context, then the bigram and trigram. The remaining series is the average number of hypotheses per frame for each utterance and is plotted on the secondary *y*-axis [3].

Table 13.6 illustrates how the depth of the ε-heap, the active states per frame of speech, word error rate, and search time change when the value of threshold ε increases for the 20,000-word WSJ dictation task. As we can see from the table, the average heap size for active nodes is only about 1.6 for the most accurate configuration. Figure 13.14(a) illustrates the distribution of stack depths for a large data set, showing that the stack depth is small even for tree initial nodes. Figure 13.14(b) illustrates the profile of the average stack depth for a sample utterance, showing that the average stack depth remains small across an utterance.

Whisper also employs look-ahead techniques to further reduce the search effort. The acoustic look-ahead technique described in Section 13.2.3.1 attempts to estimate the probability that a phonetic HMM will participate in the final result [3]. Whisper implements acoustic look-ahead by running a CI phone-net synchronously with the search process but offset N frames in the future. One side effect of the acoustic look-ahead is to provide information for the RGR strategy, as described in Section 13.2.3.2, so the search can avoid unnecessary Gaussian computation. Figure 13.15 demonstrates the effectiveness of varying the frame look-ahead from 0 to N frames in terms of states evaluated.

When the look-ahead is increased from 0 to 3 frames, the search effort, in terms of real time, is reduced by ~40% with no loss in accuracy; however, most of that is due to reducing the number of states evaluated per frame. There is no effect on the number of Gaussians evaluated per frame (the system using continuous density) until we begin to negatively impact error rate, indicating that the acoustic space represented by the pruned states is redundant and adequately covered by the retained states prior to the introduction of search errors.

With the techniques discussed here, Whisper is able to achieved real-time performance for the continuous WSJ dictation task (60,000-word) on Pentium-class PCs. The recognition accuracy is identical to that of a standard Viterbi beam decoder with a linear lexicon.

Table 13.6 Effect of heap threshold on contexts/node, states/frame-of-speech (fos), word error rate, and search time [4].

ε	Context / node	states / fos	%error	search time
0	1.000	8805	16.4	1.0x
1.0	1.001	8808	15.5	1.0x
2.0	1.008	8898	14.4	1.0x
3.0	1.018	9252	12.4	1.07x
4.0	1.056	10224	10.5	1.16x
5.0	1.147	11832	10.3	1.36x
6.0	1.315	13749	10.0	1.60x
7.0	1.528	15342	9.9	1.81x
8.0	1.647	15984	9.9	1.86x

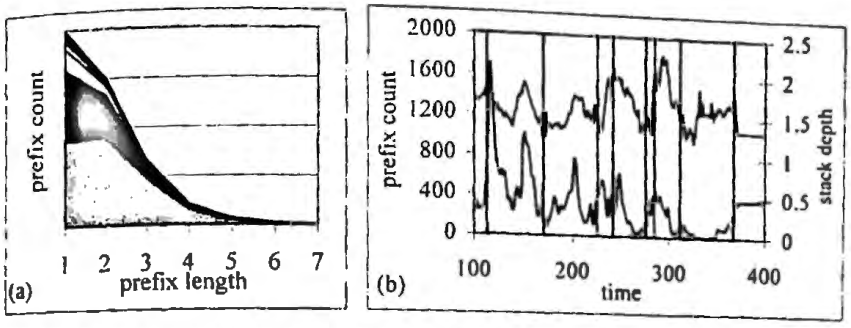


Figure 13.14 (a) A cumulative graph of the prefix count for each stack depth, starting with depth 1, showing the distribution according to prefix length. (b) The prefix count and the average stack depth with respect to one utterance. The vertical bars show the word boundaries [3].

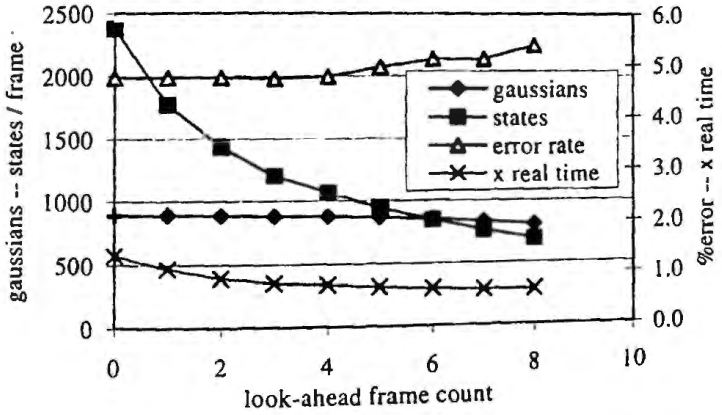


Figure 13.15 Search effort, percent error rate, and real-time factor as a function of the acoustic look-ahead. Note that search effort is the number of Gaussians evaluated per frame and the number of states evaluated per frame [3].

13.6. Historical Perspective and Further Reading

Large-vocabulary continuous speech recognition is a computationally intensive task. Real-time systems started to emerge in the late 1980s. Before that, most systems achieved real-time performance with the help of special hardware [11, 16, 25, 28]. Thanks to Moore's law and various efficient search techniques, real-time systems became a reality on a single-chip general-purpose personal computer in the 1990s [4, 34, 43].

Common wisdom in 1980s saw stack decoding as more efficient for large-vocabulary continuous speech recognition with higher-order n-grams. Time-synchronous Viterbi beam search, as described in Sections 13.1 and 13.2, emerged as the most efficient search frame-

work. It has become the most widely used search technique today. The lexical tree representation was first used by IBM as part of its allophonic fast match system [10]. Ney proposed the use of the lexical tree as the primary representation for the search space [32]. The ideas of language model factoring [4, 19] [5] and subtree polymorphism [4] enabled real-time single-pass search with higher-order language models (bigrams and trigrams). Alleva [3] and Ney [33] are two excellent articles regarding the detailed Viterbi beam search algorithm with lexical tree representation.

As mentioned in Chapter 12, fast match was first invented to speed up stack decoding [8, 9]. Ney and Ortmanns [33] and Alleva [3] extended the fast match idea to phone look-ahead in time-synchronous search by using context-independent model evaluation. In Haeb-Umbach et al. [22], a word look-ahead is implemented for a 12.3k-word speaker-dependent continuous speech recognition task. The look-ahead is performed on a lexical tree, with beam search executed every other frame. The results show a factor of 3–5 times of reduction for search space compared to the standard Viterbi beam search, while only 1–2% extra errors are introduced by word look-ahead.

The idea of multipass search strategy has long existed for knowledge-based speech recognition systems [17], where first a phone recognizer is performed, then a lexicon hypothesizer is used to locate all the possible words to form a word lattice, and finally a language model is used to search for the most possible word sequence. However, HMM's popularity predominantly shifted the focus to the unified search approach to achieve global optimization. Computation concerns led many researchers to revisit the multipass search strategy. The first n-best algorithm, described in Section 13.3.2, was published by researchers at BBN [39]. Since then, n-best and word-lattice based multipass search strategies have become important search frameworks for rapid system deployment, research tools, and spoken language understanding systems. Schwartz et al.'s paper [40] is a good tutorial on the n-best or word-lattice generation algorithms. Most of the n-best search algorithms can be made to generate word lattices/graphs with minor modifications. Other excellent discussions of multipass search can be found in [14, 24, 30].

REFERENCES

[1] Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, 1974, Addison-Wesley Publishing Company.

[2] Aho, A.V., R. Sethi, and J.D. Ullman, *Compilers: Principles, Techniques, and Tools*, 1985, Addison-Wesley.

[3] Alleva, F., "Search Organization in the Whisper Continuous Speech Recognition System," *IEEE Workshop on Automatic Speech Recognition*, 1997.

[4] Alleva, F., X. Huang, and M.Y. Hwang, "Improvements on the Pronunciation Prefix Tree Search Organization," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1996, Atlanta, Georgia, pp. 133-136.

[5] Aubert, X., *et al.*, "Large Vocabulary Continuous Speech Recognition of *Wall Street Journal* Corpus," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 129-132.

[6] Aubert, X. and H. Ney, "Large Vocabulary Continuous Speech Recognition Using Word Graphs," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 49-52.

[7] Austin, S., R. Schwartz, and P. Placeway, "The Forward-Backward Search Algorithm for Real-Time Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 697-700.

[8] Bahl, L.R., *et al.*, "Obtaining Candidate Words by Polling in a Large Vocabulary Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1988, pp. 489-492.

[9] Bahl, L.R., *et al.*, "Matrix Fast Match: a Fast Method for Identifying a Short List of Candidate Words for Decoding," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, pp. 345-347.

[10] Bahl, L.R., P.S. Gopalakrishnan, and R.L. Mercer, "Search Issues in Large Vocabulary Speech Recognition," *Proc. of the 1993 IEEE Workshop on Automatic Speech Recognition*, 1993, Snowbird, UT.

[11] Bisiani, R., T. Anantharaman, and L. Butcher, "BEAM: An Accelerator for Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, pp. 782-784.

[12] Brugnara, F. and M. Cettolo, "Improvements in Tree-Based Language Model Representation," *Proc. of the European Conf. on Speech Communication and Technology*, 1995, Madrid, Spain, pp. 1797-1800.

[13] Cettolo, M., R. Gretter, and R.D. Mori, "Knowledge Integration" in *Spoken Dialogues with Computers*, R.D. Mori, ed., Academic Press, 1998, London, pp. 231-256.

[14] Cettolo, M., R. Gretter, and R.D. Mori, "Search and Generation of Word Hypotheses" in *Spoken Dialogues with Computers*, R.D. Mori, ed., 1998, London, Academic Press, pp. 257-310.

[15] Chou, W., C.H. Lee, and B.H. Juang, "Minimum Error Rate Training Based on N-best String Models," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 652-655.

[16] Chow, Y.L., *et al.*, "BYBLOS: The BBN Continuous Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1987, pp. 89-92.

[17] Cole, R.A., *et al.*, "Feature-Based Speaker Independent Recognition of English Letters," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1983, pp. 731-734.

[18] Davenport, J.C., R. Schwartz, and L. Nguyen, "Towards A Robust Real-Time Decoder," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1999, Phoenix, Arizona, pp. 645-648.

[19] Federico, M., *et al.*, "Language Modeling for Efficient Beam-Search," *Computer Speech and Language*, 1995, pp. 353-379.

[20] Gauvain, J.L., L. Lamel, and M. Adda-Decker, "Developments in Continuous Speech Dictation using the ARPA WSJ Task," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 65-68.

[21] Gillick, L.S. and R. Roth, "A Rapid Match Algorithm for Continuous Speech Recognition," *Proc. of the Speech and Natural Language Workshop*, 1990, Hidden Valley, PA, pp. 170-172.

[22] Haeb-Umbach, R. and H. Ney, "A Look-Ahead Search Technique for Large Vocabulary Continuous Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1991, Genova, Italy, pp. 495-498.

[23] Haeb-Umbach, R. and H. Ney, "Improvements in Time-Synchronous Beam-Search for 10000-Word Continuous Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1994, 2(4), pp. 353-365.

[24] Hetherington, I.L., *et al.*, "A* Word Network Search for Continuous Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1993, Berlin, Germany, pp. 1533-1536.

[25] Hon, H.W., *A Survey of Hardware Architectures Designed for Speech Recognition*, 1991, Carnegie Mellon University, Pittsburgh, PA.

[26] Huang, X., *et al.*, "From Sphinx II to Whisper: Making Speech Recognition Usable," in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, Kluwer Academic Publishers, pp. 481-508.

[27] Huang, X., *et al.*, "Microsoft Windows Highly Intelligent Speech Recognizer: Whisper," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, pp. 93-96.

[28] Jelinek, F., "The Development of an Experimental Discrete Dictation Recognizer," *Proc. of the IEEE*, 1985, 73(1), pp. 1616-1624.

[29] Marino, J. and E. Monte, "Generation of Multiple Hypothesis in Connected Phonetic-Unit Recognition by a Modified One-Stage Dynamic Programming Algorithm," *Proc. of EuroSpeech*, 1989, Paris, pp. 408-411.

[30] Murveit, H., *et al.*, "Large Vocabulary Dictation Using SRI's DECIPHER Speech Recognition System: Progressive Search Techniques," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 319-322.

[31] Ney, H. and X. Aubert, "A Word Graph Algorithm for Large Vocabulary," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 1355-1358.

[32] Ney, H., *et al.*, "Improvements in Beam Search for 10000-Word Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, California, pp. 9-12.

[33] Ney, H. and S. Ortmanns, *Dynamic Programming Search for Continuous Speech Recognition*, in *IEEE Signal Processing Magazine*, 1999, pp. 64-83.

[34] Nguyen, L., *et al.*, "Search Algorithms for Software-Only Real-Time Recognition with Very Large Vocabularies," *Proc. of ARPA Human Language Technology Workshop*, 1993, Plainsboro, NJ, pp. 91-95.

[35] Nilsson, N.J., *Problem-Solving Methods in Artificial Intelligence*, 1971, New York, McGraw-Hill.

[36] Normandin, Y., "Maximum Mutual Information Estimation of Hidden Markov Models" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, Kluwer Academic Publishers.

[37] Odell, J.J., *et al.*, "A One Pass Decoder Design for Large Vocabulary Recognition," *Proc. of the ARPA Human Language Technology Workshop*, 1994, Plainsboro, NJ, pp. 380-385.

[38] Schwartz, R. and S. Austin, "A Comparison of Several Approximate Algorithms for Finding Multiple (N-BEST) Sentence Hypotheses," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 701-704.

[39] Schwartz, R. and Y.L. Chow, "The N-Best Algorithm: an Efficient and Exact Procedure for Finding the N Most Likely Sentence Hypotheses," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1990, Albuquerque, New Mexico, pp. 81-84.

[40] Schwartz, R., L. Nguyen, and J. Makhoul, "Multiple-Pass Search Strategies" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Norwell, MA, Klewer Academic Publishers, pp. 57-81.

[41] Soong, F.K. and E.F. Huang, "A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 705-708.

[42] Steinbiss, V., "Sentence-Hypotheses Generation in a Continuous Speech Recognition," *Proc. of EuroSpeech*, 1989, Paris, pp. 51-54.

[43] Steinbiss, V., *et al.*, "The Philips Research System for Large-Vocabulary Continuous-Speech Recognition," *Proc. of the European Conf. on Speech Communication and Technology*, 1993, Berlin, Germany, pp. 2125-2128.

[44] Woodland, P.C., *et al.*, "Large Vocabulary Continuous Speech Recognition Using HTK," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 125-128.

PART IV

TEXT-TO-SPEECH SYSTEMS

CHAPTER 14

Text and Phonetic Analysis

Text-to-speech can be viewed as a speech coding system that yields an extremely high compression ratio coupled with a high degree of flexibility in choosing style, voice, rate, pitch range, and other playback effects. In this view of TTS, the text file that is input to a speech synthesizer is a form of coded speech. Thus, TTS subsumes coding technologies discussed in Chapter 7 with the following goals:

- *Compression ratios superior to digitized wave files*—Compression yields benefits in many areas, including fast Internet transmission of spoken messages.
- *Flexibility in output characteristics*—Flexibility includes easy change of gender, average pitch, pitch range, etc., enabling application developers to give their systems' spoken output a unique individual personality. Flexibility also implies easy change of message content; it is generally easier to retype text than it is to record and deploy a digitized speech file.
- *Ability for perfect indexing between text and speech forms*—Preservation of the correspondence between textual representation and the speech wave form allows synchronization with other media and output modes, such as word-by-word reverse video highlighting in a literacy tutor reading aloud.

- *Alternative access of text content*—TTS is the most effective alternative access of text for the blind, hands-free/eyes-free and displayless scenarios.

At first sight, the process of converting text into speech looks straightforward. However, when we analyze how complicated speakers read a text aloud, this simplistic view quickly falls apart. First, we need to convert words in written forms into speakable forms. This process is clearly nontrivial. Second, to sound natural, the system needs to convey the intonation of the sentences properly. This second process is clearly an extremely challenging one. One good analogy is to think how difficult it is to drop a foreign accent when speaking a second language—a process still not quite understood by human beings.

The ultimate goal of simulating the speech of an understanding, effective human speaker from plain text is as distant today as the corresponding Holy Grail goals of the fields of speech recognition and machine translation. This is because such humanlike rendition depends on common-sense reasoning about the world and the text's relation to it, deep knowledge of the language itself in all its richness and variability, and even knowledge of the actual or expected audience—its goals, assumptions, presuppositions, and so on. In typical audio books or recordings for the visually challenged today, the human reader has enough familiarity with and understanding of the text to make appropriate choices for rendition of emotion, emphasis, and pacing, as well as handling both dialog and exposition. While computational power is steadily increasing, there remains a substantial knowledge gap that must be closed before fully human-sounding simulated voices and renditions can be created.

While no TTS system to date has approached optimal quality in the Turing test,[1] a large number of experimental and commercial systems have yielded fascinating insights. Even the relatively limited-quality TTS systems of today have found practical applications.

The basic TTS system architecture is illustrated in Chapter 1. In the present chapter we discuss text analysis and phonetic analysis whose objective is to convert words into speakable phonetic representation. The techniques discussed here are relevant to what we discussed for language modeling in Chapter 11 (like text normalization before computing *n*-gram) and for pronunciation modeling in Chapter 9. The next two modules—prosodic analysis and speech synthesis—are treated in the next two chapters.

14.1. MODULES AND DATA FLOW

The text analysis component, guided by presenter controls, is typically responsible for determining document structure, conversion of nonorthographic symbols, and parsing of language structure and meaning. The phonetic analysis component converts orthographic words to phones (unambiguous speech sound symbols). Some TTS systems assume dependency between text analysis, phonetic analysis, prosodic analysis, and speech synthesis, particularly systems based on very large databases containing long stretches of original, unmodified

[1] A test proposed by British mathematician Allan Turing of the ability of a computer to flawlessly imitate human performance on a given speech or language task [29].

digitized speech with their original pitch contours. We discuss our high-level linguistic description of those modules, based on modularity, transparency, and reusability of components, although some aspects of text and phonetic analysis may be unnecessary for some particular systems.

We assume that the entire text (word, sentence, paragraph, document) to be spoken is contained in a single, wholly visible buffer. Some systems may be faced with special requirements for continuous flow-through or visibility of only small (word, phrase, sentence) *chunks* at a time, or extremely complex timing and synchronization requirements. The basic functional processes within the text and phonetic analysis are shown schematically in Figure 14.1.

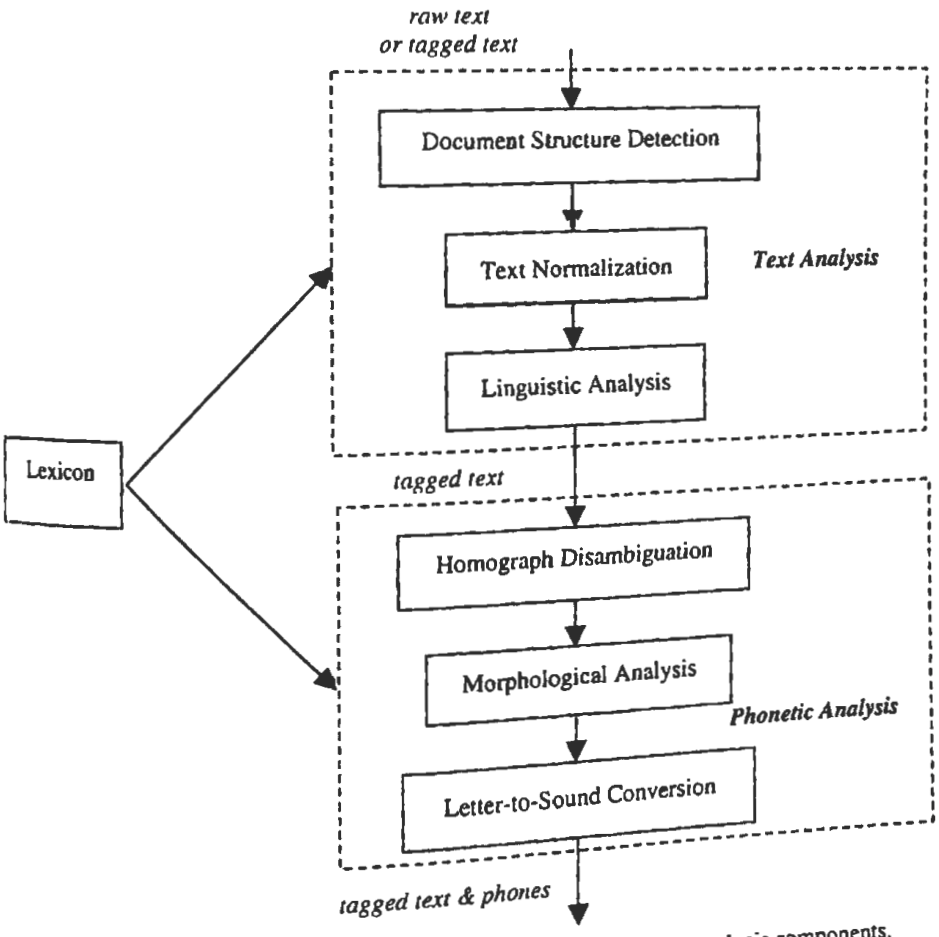


Figure 14.1 Modularized functional blocks for text and phonetic analysis components.

The architecture in Figure 14.1 brings the standard benefits of modularity and transparency. Modularity in this case means that the analysis at each level can be supplied by the most expert knowledge source, or a variety of different sources, as long as the markup conventions for expressing the analysis are uniform. Transparency means that the results of each stage could be reused by other processes for other purposes.

14.1.1. Modules

The *text analysis module* (TAM) is responsible for indicating all knowledge about the text or message that is not specifically phonetic or prosodic in nature. Very simple systems do little more than convert nonorthographic items, such as numbers, into words. More ambitious systems attempt to analyze whitespaces and punctuations to determine document structure, and perform sophisticated syntax and semantic analysis on sentences to determine attributes that help the phonetic analysis to generate correct phonetic representation and prosodic generation to construct superior pitch contours. As shown in Figure 14.1, text analysis for TTS involves three related processes:

- *Document structure detection*—Document structure is important to provide a context for all later processes. In addition, some elements of document structure, such as sentence breaking and paragraph segmentation, may have direct implications for prosody.
- *Text normalization*—Text normalization is the conversion from the variety of symbols, numbers, and other nonorthographic entities of text into a common orthographic transcription suitable for subsequent phonetic conversion.
- *Linguistic analysis*—Linguistic analysis recovers the syntactic constituency and semantic features of words, phrases, clauses, and sentences, which is important for both pronunciation and prosodic choices in the successive processes.

The task of the phonetic analysis is to convert lexical orthographic symbols to phonemic representation along with possible diacritic information, such as stress placement. Phonetic analysis is thus often referred to as grapheme-to-phoneme conversion. The purpose is obvious, since phonemes are the basic units of sound, as described in Chapter 2. Even though future TTS systems might be based on word sounding units with increasing storage technologies, homograph disambiguation and phonetic analysis for new words (either true new words being invented over time or morphologically transformed words) are still necessary for systems to correctly utter every word.

Grapheme-to-phoneme conversion is trivial for languages where there is a simple relationship between orthography and phonology. Such a simple relationship can be well captured by a handful of rules. Languages such as Spanish and Finnish belong to this category and are referred to as *phonetic languages*. English, on the other hand, is remote from pho-

netic language because English words often have many distinct origins. It is generally believed that the following three services are necessary to produce accurate pronunciations.

- *Homograph disambiguation*—It is important to disambiguate words with different senses to determine proper phonetic pronunciations, such as object (*/ah b jh eh k t/*) as a verb or as a noun (*/aa b jh eh k t/*).
- *Morphological analysis*—Analyzing the component morphemes provides important cues to attain the pronunciations for inflectional and derivational words.
- *Letter-to-sound conversion*—The last stage of the phonetic analysis generally includes general letter-to-sound rules (or modules) and a dictionary lookup to produce accurate pronunciations for any arbitrary word.

All the processes in text and phonetic analysis phases above need not to be deterministic, although most TTS systems today have deterministic processes. What we mean by *not deterministic* is that each of the above processes can generate multiple hypotheses with the hope that the later process can disambiguate those hypotheses by using more knowledge. For example, often it might not be trivial to decide whether the punctuation "." is a sentence ending mark or abbreviation mark during document structure detection. The document structure detection process can pass both hypotheses to the later processes, and the decision can then be delayed until there is enough information to make an informed decision in later modules, such as the text normalization or linguistic analysis phases. When generating multiple hypotheses, the process can also assign probabilistic information if it comprehends the underlying probabilistic structure. This flexible pipeline architecture avoids the mistakes made by early processes based on insufficient knowledge.

Much of the work done by the text/phonetic analysis phase of a TTS system mirrors the processing attempted by *natural language process* (NLP) systems for other purposes, such as automatic proofreading, machine translation, database document indexing, and so on. Increasingly sophisticated NL analysis is needed to make certain TTS processing decisions in the examples illustrated in Table 14.1. Ultimately all decisions are context driven and probabilistic in nature, since, for example, dogs might be cooked and eaten in some cultures.

Table 14.1 Examples of several ambiguous text normalization cases.

Examples	Alternatives	Techniques
Dr. Smith	*doctor* or *drive*?	abbreviation analysis, case analysis
Will you go?	yes-no or wh-question?	syntactic analysis
I ate a hot dog.	accent on *dog*?	semantic, verb/direct object likelihood
I saw a hot dog.	accent on *dog*?	discourse, pragmatic analysis

Most TTS systems today employ specialized natural language processing modules for front-end analysis. In the future, it is likely that less emphasis will be placed on construction of TTS-specific text/phonetic analysis components such as those described in [27], while more resources will likely go to general-purpose NLP systems with cross-functional potential [23]. In other words, all the modules above only perform simple processing and pass all possible hypotheses to the later modules. At the end of the text/phonetic phase, a unified NLP module then performs extensive syntactic/semantic analysis for the best decisions. The necessity for such an architectural approach is already visible in markets where language issues have forced early attention to common lexical and tokenization resources, such as Japan. Japanese system services and applications can usually expect to rely on common cross-functional linguistic resources, and many benefits are reaped, including elimination of bulk, reduction of redundancy and development time, and enforcement of systemwide consistent behavior. For example, under Japanese architectures, TTS, recognition, sorting, word processing, database, and other systems are expected to share a common language and dictionary service.

14.1.2. Data Flows

It is arguable that text input alone does not give the system enough information to express and render the intention of the text producer. Thus, more and more TTS systems focus on providing an infrastructure of standard set of markups (tags), so that the text producer can better express their semantic intention with these markups in addition to plain text. These kinds of markups have different levels of granularity, ranging from simple speed settings specified in *words per minute* up to elaborate schemes for semantic representation of concepts that may bypass the ordinary text analysis module altogether.[2] The markup can be done by internal proprietary conventions or by some standard markup, such as XML (Extensible Markup Language [35]). Some of these markup capabilities will be discussed in Sections 14.3 and 14.4.

For example, an application may know a lot about the structure and content of the text to be spoken, and it can apply this knowledge to the text, using common markup conventions, to greatly improve spoken output quality. On the other hand, some applications may have certain broad requirements such as rate, pitch, callback types, etc. For engines providing such supports, the text and/or phonetic analysis phase can be skipped, in whole or in part. Whether the application or the system has provided the text analysis markup, the structural conventions should be identical and must be sufficient to guide the phonetic analysis. The phonetic analysis module should be presented only with markup tags indicating structure or functions of textual chunks, and words in standard orthography. The similar phonetic markups could also be presented to the phonetic analysis module, the module could be skipped.

[2] This latter type of system is sometimes called *concept-to-speech* or *message-to-speech*, which is described in Chapter 17. It generally generates better speech rendering when domain-specific knowledge is provided to the system.

Internal architectures, data structures, and interfaces may vary widely from system to system. However, most modern TTS systems initially construct a simple description of an utterance or paragraph based on observable attributes, typically text words and punctuation, perhaps augmented by control annotations. This minimal initial *skeleton* is then augmented with many layers of structure hypothesized by the TTS system's internal analysis modules. Beginning with a surface stream of words, punctuation, and other symbols, typical layers of detected structure that may be added include:

- Phonemes
- Syllables
- Morphemes
- Words derived from nonwords (such as dates like "9/10/99")
- Syntactic constituents
- Relative importance of words and phrases
- Prosodic phrasing
- Accentuation
- Duration controls
- Pitch controls

We can now consider how the information needed to support synthesis of a sentence is developed in processing an example sentence such as: "A skilled electrician reported."

In Figure 14.2, the information that must be inferred from text is diagrammed. The flow proceeds as follows:

- **W(ords) → Σ, C(ontrols)**: the syllabic structure (Σ) and the basic phonemic form of a word are derived from lexical lookup and/or the application of rules. The Σ tier shows the syllable divisions (written in text form for convenience). The C tier, at this stage, shows the basic phonemic symbols for each word's syllables.
- **W(ords) → S(yntax/semantics)**: The word stream from text is used to infer a syntactic and possibly semantic structure (S tier) for an input sentence. Syntactic and semantic structure above the word would include syntactic constituents such as Noun Phrase (NP), Verb Phrase (VP), etc. and any semantic features that can be recovered from the current sentence or analysis of other contexts that may be available (such as an entire paragraph or document). The lower-level phrases such as NP and VP may be grouped into broader constituents such as Sentence (S), depending on the parsing architecture.
- **S(yntax/semantics) → P(rosody)**: The P(rosodic) tier is also called the symbolic prosodic module. If a word is semantically *important* in a sentence, that importance can be reflected in speech with a little extra phonetic prominence, called an accent. Some synthesizers begin building a prosodic structure by

placing metrical foot boundaries to the left of every accented syllable. The resulting metrical foot structure is shown as F1, F2, etc. in Figure 14.2 (some *feet* lack an accented head and are 'degenerate'). Over the metrical foot structure, higher-order prosodic constituents, with their own characteristic relative pitch ranges, boundary pitch movements, etc. can be constructed, shown in the figure as intonational phrases IP1, IP2. The details of prosodic analysis, including the meaning of those symbols, are described in Chapter 15.

The final phonetic form of the words to be spoken will reflect not only the original phonetics, but decisions made in the S and P tiers as well. For example, the P(rosody) tier adds detailed pitch and duration controls to the C(ontrol) specification that is passed to the voice synthesis component. Obviously, there can be a huge variety of particular architectures and components involved in the conversion process. Most systems, however, have some analog to each of the components presented above.

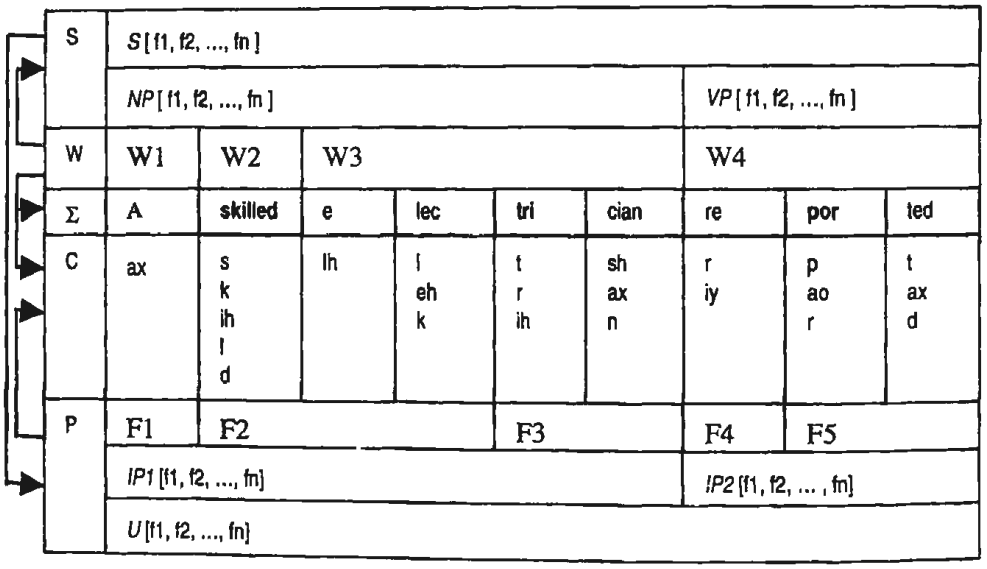


Figure 14.2 Annotation tiers indicating incremental analysis based on an input (text) sentence "A skilled electrician reported." Flow of incremental annotation is indicated by arrows on the left side.

14.1.3. Localization Issues

A major issue in the text and phonetic analysis components of a TTS system is internationalization and localization. While most of the language processing technologies in this book are exemplified by English case studies, an internationalized TTS architecture enabling minimal expense in localization is highly desirable. From a technological point of view, the text conventions and writing systems of language communities may differ substantially in arbitrary ways, necessitating serious effort in both specifying an internationalized architec-

ture for text and phonetic analysis, and localizing that architecture for any particular language.

For example, in Japanese and Chinese, the unit of *word* is not clearly identified by spaces in text. In French, interword dependencies in pronunciation realization exist (liaison). Conventions for writing numerical forms of dates, times, money, etc. may differ across languages. In French, number groups separated by spaces may need to be integrated as single amounts, which rarely occurs in English. Some of these issues may be more serious for certain types of TTS architectures than others. In general, it is best to specify a rule architecture for text processing and phonetic analysis based on some fundamental formalism that allows for language-particular data tables, and which is powerful enough to handle a wide range of relations and alternatives.

14.2. LEXICON

The most important resource for text and phonetic analysis is the TTS system lexicon (also referred to as a dictionary). As illustrated in Figure 14.1, the TTS system lexicon is shared with almost all components. The lexical service should provide the following kinds of content in order to support a TTS system:

- Inflected forms of lexicon entries
- Phonetic pronunciations (support multiple pronunciations), stress and syllabic structure features for each lexicon entry
- Morphological analysis capability
- Abbreviation and acronym expansion and pronunciation
- Attributes indicating word status, including proper-name tagging, and other special properties
- List of speakable names of all common single characters. Under modern operating systems, the characters should include all Unicode characters.
- Word part-of-speech (POS) and other syntactic/semantic attributes
- Other special features, e.g., how likely a word is to be accented, etc.

It should be clear that the requirements for a TTS system lexical service overlap heavily with those for more general-purpose NLP.

Traditionally, TTS systems have been rule oriented, in particular for grapheme-to-phoneme conversion. Often, tens of so called *letter-to-sound* (LTS) rules (described in detail in Section 14.8) are used first for grapheme-to-phoneme conversion, and the role of the lexicon has been minimized as an *exception list*, whose pronunciations cannot be predicted on the basis of such LTS rules. However, this view of the lexicon's role has increasingly been adjusted as the requirement of a sophisticated NLP analysis for high-quality TTS systems has become apparent. There are a number of ways to optimize a dictionary system. For a good overview of lexical organization issues, please see [4].

To expose different contents about a lexicon entry listed above for different TTS module, it calls for a consistent mechanism. It can be done either through a database query or a function call in which the caller sends a key (usually the orthographic representation of a word) and the desired attribute. For example, a TTS module can use the following function call to look up a particular attribute (like phonetic pronunciations or POS) by passing the attribute *att* and the result will be stored in the pointer *val* upon successful lookup. Moreover, when the lookup is successful (the word is found in the dictionary) the function returns true, otherwise it will return false instead.

```
BOOLEAN DictLookup (string word, ATTTYPE att, (VOID *) val)
```

We should also point out that this functional view of dictionary could further expand the physical dictionary as a service. The morphological analysis and letter-to-sound modules (described in Sections 14.7 and 14.8) can all be incorporated into the same lexical service. That is, underneath dictionary lookup, operation and analysis is encapsulated from users to form a uniform service.

Another consideration in the system's runtime dictionary is compression. While many standard compression algorithms exist, and should be judiciously applied, the organization and extent of the vocabulary itself can also be optimized for small space and quick search. The kinds of American English vocabulary relevant to a TTS system include:

- Grammatical function words (closed class)—about several hundred
- Very common vocabulary—about 5,000 or more
- College-level core vocabulary base forms—about 60,000 or more
- College-level core vocabulary inflected form—about 120,000 or more
- Scientific and technical vocabulary, by field—e.g., legal, medical, engineering, etc.
- Personal names—e.g., family, given, male, female, national origin, etc.
- Place names—e.g., countries, cities, rivers, mountains, planets, stars, etc.
- Slang
- Archaisms

The typical sizes of reasonably complete lists of the above types of vocabulary run from a few hundred function or closed-class words (such as prepositions and pronouns) to 120,000 or so inflected forms of college-level vocabulary items, up to several million surnames and place names. Careful analysis of the likely needs of typical target applications can potentially reduce the size of the runtime dictionary. In general, most TTS systems maintain a system dictionary with a size between 5000 and 200,000 entries. With advanced technologies in database and hashing, search is typically a nonissue for dictionary lookup. In addition, since new forms are constantly produced by various creative processes, such as acronyms, borrowing, slang acceptance, compounding, and morphological manipulation, some means of analyzing words that have not been stored must be provided. This is the topic of Sections 14.7 and 14.8.

14.3. DOCUMENT STRUCTURE DETECTION

For the purpose of discussion, we assume that all input to the TAM is an XML document, though perhaps largely unmarked, and the output is also a (more extensively marked) XML document. That is to say, all the knowledge recovered during the TAM phase is to be expressed as XML markup. This confirms the independence of the TAM from phonetic and prosodic considerations, allowing a variety of resources, some perhaps not crafted with TTS in mind, to be brought to bear by the TAM on the text. It also implies that that output of the TAM is potentially usable by other, non-TTS processes, such as normalization of language-model training data for building statistical language models (see Chapter 11). This fully modular and transparent view of TTS allows the greatest flexibility in document analysis, provides for direct *authoring* of structure and other customization, while allowing a split between expensive, multipurpose natural language analysis and the core TTS functionality. Although other text format or markup language, such as Adobe Acrobat or Microsoft Word, can be used for the same purpose, the choice of XML is obvious because it is the widely open standard, particularly for the Internet.

XML is a set of conventions for indicating the semantics and scope of various entities that combine to constitute a document. It is conceptually somewhat similar to Hypertext Markup Language (HTML), which is the exchange code for the World Wide Web. In these markup systems, properties are identified by tags with explicit scope, such as `"<b>make this phrase bold</b>"` to indicate a heavy, dark print display. XML in particular attempts to enforce a principled separation between document structure and content, on one hand, and the detailed formatting or presentation requirements of various *uses* of documents, on the other. Since we cannot provide a tutorial on XML here, we freely introduce example tags that indicate document and linguistic structure. The interpretations of these are intuitive to most readers, though, of course, the analytic knowledge underlying decisions to insert tags may be very sophisticated. It will be some time before commercial TTS engines come to a common understanding on the wide variety of text attributes that should be marked, and accept a common set of conventions. Nevertheless, it is reasonable to adopt the idea that TAM should be independent and reusable, thus allowing XML documents (which are expected to proliferate) to function for speech just as for other modalities, as indicated schematically in Figure 14.3.

TTS is regarded in Figure 14.3 as a factored process, with the text analysis perhaps carried out by human editors or by natural language analysis systems. The role of the TTS engine per se may eventually be reduced to the interpretation of structural tags and provision of phonetic information. While commercial engines of the present day are not structured with these assumptions in mind, modularity and transparency are likely to become increasingly important. The increasing acceptance of the basic ideas underlying an XML documentcentric approach to text and phonetic analysis for TTS can be seen in the recent proliferation of XML-like speech markup proposals [24, 33]. While not presenting any of these in detail, in the discussion below we adopt informal conventions that reflect and extend their basic assumptions. The structural markup exploited by the TTS systems of the

future may be imposed by XML authoring systems at document creation time, or may be inserted by independent analytical procedures. In any case the distinction between purely automatic structure creation/detection and human annotation and authoring will increasingly blur—just as in natural language translation and information retrieval domains, the distinction between machine-produced results and human-produced results has begun to blur.

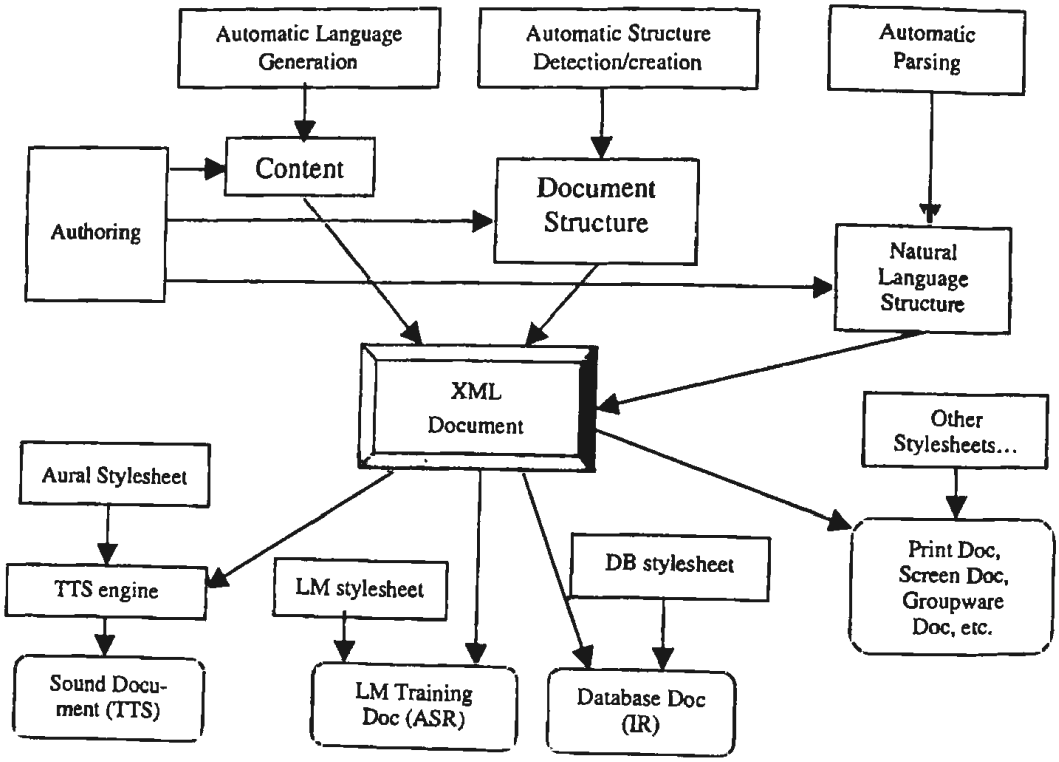


Figure 14.3 A documentcentric view of TTS.

14.3.1. Chapter and Section Headers

Section headers are a standard convention in XML document markup, and TTS systems can use the structural indications to control prosody and to regulate prosodic style, just as a professional reader might treat chapter headings differently. Increasingly, a document created on computer or intended for any kind of electronic circulation incorporates structural markup, and the TTS and audio human-computer-interface systems of the future learn to exploit this (in longer documents, the document structure markup assists in audio navigation, speedup, and skipping). For example, the XML annotation of a book at a high level might follow conventions as shown in Figure 14.4. Viewing a document in this way might lead a TTS system to insert pauses and emphasis correctly, in accordance with the structure

marked. Furthermore, an audio interface system would work jointly with a TTS system to allow easy navigation and orientation within such a structure. If future documents are marked up in this fashion, the concept of audio books, for example, would change to rely less on unstructured prerecorded speech and more on smart, XML-aware, high-quality audio navigation and TTS systems, with the output customization flexibility they provide.

For documents without explicit markup information for section and chapter headers, it is in general a nontrivial task to detect them automatically. Therefore, most TTS systems today do not make such an attempt.

```
<Book>
   <Title>The Pity of War</Title>
    <Subtitle>Explaining World War I</Subtitle>
   <Author>Niall Ferguson</Author>
   <TableOfContents>...</TableOfContents>
   <Introduction>
         <Para>...</Para>
         ...
   </Introduction>
   <Chapter>
   <ChapterTitle>The Myths of Militarism</ChapterTitle>
         <Section>
               <SectionTitle>Prophets</SectionTitle>
               <Para> ... </Para>
               ...
         </Section>
    </Chapter>
    ...
</Book>
```

Figure 14.4 An example of the XML annotation of a book.

14.3.2. Lists

Lists or bulleted items may be rendered with distinct intonational contours to indicate aurally their special status. This kind of structure might be indicated in XML as shown in Figure 14.5. Again, TTS engine designers need to get used to the idea of accepting such markup for interpretation, or incorporating technologies that can detect and insert such markup as needed by the downstream phonetic processing modules. Similar to chapter and section headers, most TTS systems today do not make an attempt to detect list structures automatically.

```
<UL>
<LI>compression</LI>
<LI>flexibility</LI>
<LI>text-waveform correspondence</LI>
</UL>
<Caption>The advantages of TTS</Caption>
```

Figure 14.5 An example of a list marked by XML.

14.3.3. Paragraphs

The paragraph has been shown to have direct and distinctive implications for pitch assignment in TTS [26]. The pitch range of good readers or speakers in the first few clauses at the start of a new paragraph is typically substantially higher than that for mid-paragraph sentences, and it narrows further in the final few clauses, before resetting for the next paragraph. Thus, to mimic a high-quality reading style in future TTS systems, the paragraph structure has to be detected from XML tagging or inferred from inspection of raw formatting. Obviously, relying on independently motivated XML tagging is, as always, the superior option, especially since this is a very common structural annotation in XML documents.

In contrast to other document structure information, paragraphs are probably among the easiest to detect automatically. The character <CR> (carriage return) or <NL> (new line) is usually a reliable clue for paragraphs.

14.3.4. Sentences

While sentence breaks are not normally indicated in XML markup today, there is no reason to exclude them, and knowledge of the sentence unit can be crucial for high-quality TTS. In fact, some XML-like conventions for text markup of documents to be rendered by synthesizers (e.g., SABLE) provide for a DIV (division) tag that could take paragraph, sentence, clause, etc. as attribute [24]. If we define sentence broadly as a primal linguistic unit that makes up paragraphs, attributes could be added to a `Sent` tag to express whatever linguistic knowledge exists about the type of the sentence as a whole:

```
<Sent type="yes-no question">
Is life so dear, or peace so sweet, as to be purchased at the price of chains and slavery?
</Sent>
```

Again, as emphasized throughout this section, such annotation could be either applied during creation of the XML documents (of the future) or inserted by independent processes. Such structure-detection processes may be motivated by a variety of needs and may exist outside the TTS system per se.

If no independent markup of sentence structure is available from an external, independently motivated document analysis or natural language system, a TTS system typically relies on simple internal heuristics to guess at sentence divisions. In email and other relatively informal written communications, sentence boundaries may be very hard to detect. In contrast to English, sentence breaking could be trivial for some other written languages. In Chinese, there is a designated symbol (a small circle ◦) for marking the end of a sentence, so the sentence breaking could be done in a totally straightforward way. However, for most Asian languages, such as Chinese, Japanese, and Thai, there is in general no space within a sentence. Thus, tokenization is an important issue for Asian languages.

In more formal English writing, sentence boundaries are often signaled by terminal punctuation from the set: { . ! ? } followed by whitespaces and an upper-case initial word. Sometimes additional punctuation may trail the '?' and '!' characters, such as close quotation marks and/or close parenthesis. The character '.' is particularly troubling, because it is, in programming terms, heavily *overloaded*. Apart from its uses in numerical expressions and Internet addresses, its other main use is as a marker of abbreviation, itself a difficult problem for text normalization (see Section 14.4). Consider this pathological jumble of potentially ambiguous cases:

> Mr. Smith came by. He knows that it costs $1.99, but I don't know when he'll be back (he didn't ask, "when should I return?")... His Web site is www.mrsmithhhhhh.com. The car is 72.5 in. long (we don't know which parking space he'll put his car in.) but he said "...and the truth shall set you free," an interesting quote.

Some of these can be resolved in the linguistic analysis module. However for some cases, only probabilistic guesses can be made, and even a human reader may have difficulty. The ambiguous sentence breaking can also be resolved in an abbreviation-processing module (described in Section 14.4.1). Any period punctuation that is not taken to signal an abbreviation and is not part of a number can be taken as end-of-sentence. Of course, as we have seen above, abbreviations are also confusable with words that can naturally end sentences, e.g., "*in*." For the measure abbreviations, an examination of the left context (checking for numeric) may be sufficient. In any case, the complexity of sentence breaking illustrates the value of passing multiple hypotheses and letting later, more knowledgeable modules (such as an abbreviation or linguistic analysis module) make decisions. Algorithm 14.1 shows a simple sentence-breaking algorithm that should be able to handle most cases.

For advanced sentence breakers, a weighted combination of the following kinds of considerations may be used in constructing algorithms for determining sentence boundaries (ordered from easiest/most common to most sophisticated):

- Abbreviation processing—Abbreviation processing is one of the most important tasks in text normalization and will be described in detail in Section 14.4.
- Rules or CART built (Chapter 4) upon features based on: document structure, whitespace, case conventions, etc.
- Statistical frequencies on sentence-initial word likelihood

- Statistical frequencies of typical lengths of sentences for various genres
- Streaming syntactic/semantic (linguistic) analysis—Syntactic/semantic analysis is also essential for providing critical information for phonetic and prosodic analysis. Linguistic analysis will be described in Section 14.5.

As you can see, a deliberate sentence breaking requires a fair amount of linguistic processing, like abbreviation processing and syntactic/semantic analysis. Since this type of analysis is typically included in the later modules (text normalization or linguistic analysis), it might be a sensible decision to delay the decision for sentence breaking until later modules, either text normalization or linguistic analysis. In effect, this arrangement can be treated as the document structure module passing along multiple hypotheses of sentence boundaries, and it allows later modules with deeper linguistic knowledge (text normalization or linguistic analysis) to make more intelligent decisions.

Finally, if a long buffer of unpunctuated words is presented, TTS systems may impose arbitrary limits on the length of a sentence for later processing. For example, the writings of the French author Marcel Proust contain some sentences that are several hundred words long (average sentence length for ordinary prose is about 15 to 25 words).

ALGORITHM 14.1: *A SIMPLE SENTENCE-BREAKING ALGORITHM*

1. If found punctuation ./!/? advance one character and **goto 2.**
   else advance one character and **goto 1.**
2. If not found whitespace advance one character and **goto 1.**
3. If the character is period (.) **goto 4.**
   else goto 5.
4. Perform abbreviation analysis.
   If not an abbreviation **goto 5.**
   else advance one character and **goto 1.**
5. Declare a sentence boundary and sentence type ./!/?
   Advance one character and **goto 1.**

14.3.5. Email

TTS could be ideal for reading email over the phone or in an eyes-busy situation such as when driving a motor vehicle. Here again we can speculate that XML-tagged email structure, minimally something like the example in Figure 14.6, will be essential for high-quality prosody, and for controlling the audio interface, allowing skips and speedups of areas the user has defined as less critical, and allowing the system to announce the function of each block. For example, the `sig` (signature) portion of email certainly has a different semantic function than the main message text and should be clearly identified as such, or skipped, at the listener's discretion. Modern email systems are providing increasingly sophisticated

support for structure annotation such as that exemplified in Figure 14.6. Obviously, the email document structure can be detected only with appropriate tags (like XML). It is very difficult for a TTS system to detect it automatically.

```
<message>
  <header>
      <date>11 June 1998</date>
      <from>Leslie</from>
      <to>Jo</to>
      <subject>Surf's Up!</subject>
  </header>
  <body> ... </body>
  <sig>Freedom's just another word for nothing left to lose</sig>
</message>
```

Figure 14.6 An example of email marked by XML.

14.3.6. Web Pages

All the comments about TTS reliance on XML markup of document structure can be applied to the case of HTML-marked Web page content as well. In addition to sections, headers, lists, paragraphs, etc., the TTS systems should be aware of XML/HTML conventions such as links (link name) and perhaps apply some distinctive voice quality or prosodic pitch contour to highlight these. The size and color of the section of text also provides useful hints for emphasis. Moreover, the TTS system should also integrate the rendering of audio and video contents on the Web page to create a genuine multimedia experience for the users. More could be said about the rendition of Web content, whether from underlying XML documents or HTML-marked documents prepared specifically for Web presentation. In addition, the World Wide Web Consortium has begun work on standards for aural stylesheets that can work in conjunction with standard HTML to provide special direction in aural rendition [33].

14.3.7. Dialog Turns and Speech Acts

Not all text to be rendered by a TTS system is standard written prose. The more expressive TTS systems could be tasked with rendering natural conversation and dialog in a spontaneous style. As with written documents, the TTS system has to be guided by XML markup of its input. Various systems for marking *dialog turns* (change of speaker) and *speech acts* (the mood and functional intent of an utterance)[3] are used for this purpose, and these annotations will trigger particular phonetic and prosodic rules in TTS systems. The speech act coding

[3] Dialog modeling and the concepts of *dialog turns* and *speech acts* are described in detail in Chapter 17.

schemes can help, for example, in identifying the speaker's intent with respect to an utterance, as opposed to the utterance's structural attributes. The prosodic contour and voice quality selected by the TTS system might be highly dependent on this functional knowledge.

For example, a syntactically well-formed question might be used as information solicitation, with the typical utterance-final pitch upturn as shown in the following:

<REQUEST_INFO>Can you hand me the wrench?</REQUEST_INFO>

But if the same utterance is used as a command, the prosody may change drastically.

<DIRECTIVE>Can you hand me the wrench.</DIRECTIVE>

Research on speech act markup-tag inventories (see Chapter 17) and automatic methods for speech act annotation of dialog is ongoing, and this research has the property considered desirable here, in that it is independently motivated (useful for enhancing speech recognition and language understanding systems). Thus, an advanced TTS system should be expected to exploit dialog and speech act markups extensively.

14.4. TEXT NORMALIZATION

Text often include abbreviations (e.g., FDA for *Food and Drug Administration*) and acronyms (SWAT for *Special Weapons And Tactics*). Novels and short stories may include *spoken* dialog interspersed with exposition; technical manuals may include mathematical formulae, graphs, figures, charts and tables, with associated captions and numbers; email may require interpretation of special conventional symbols such as *emoticons* [e.g., :-) means smileys], as well as Web and Internet address formats, and special abbreviations (e.g., IMHO means *in my humble opinion*). Again, any text source may include part numbers, stock quotes, dates, times, money and currency, and mathematical expressions, as well as standard ordinal and cardinal formats. Without context analysis or prior knowledge, even a human reader would sometimes be hard pressed to give a perfect rendition of every sequence of nonalphabetic characters or of every abbreviation. *Text normalization* (TN) is the process of generating normalized orthography (or, for some systems, direct generation of phones) from text containing words, numbers, punctuation, and other symbols. For example, a simple example is given as follows:

The 7% Solution ➔ THE SEVEN PER CENT SOLUTION

Text normalization is an essential requirement not only for TTS, but also for the preparation of training text corpora for acoustic-model and language-model construction.[4] In addition, speech dictation systems face an analogous problem of *inverse* text normalization for document creation from recognized words, and such systems may depend on knowledge sources similar to those described in this section. The example of an inverse text normalization for the example above is given as follows:

[4] For details of acoustic and language modeling, please refer to Chapters 9 and 11.

THE SEVEN PER CENT SOLUTION → The 7% Solution

Modular text normalization components, which may produce output for multiple downstream consumers, mark up the exemplary text along the following lines:

The <tn snor="SEVEN PER CENT">7%</tn> Solution

The `snor` tag stands for *Standard Normalized Orthographic Representation.*[5] For TTS, input text may include multisentence paragraphs, numbers, dates, times, punctuation, symbols of all kinds, as well as interpretive annotations in a TTS markup language, such as tags for word emphasis or pitch range. Text analysis for TTS is the work of converting such text into a stream of normalized orthography, with all relevant input tagging preserved and new markup added to guide the subsequent modules. Such interpretive annotations added by text analysis are critical for phonetic and prosodic generation phases to produce desired output. The output of the text normalizer may be deterministic, or may preserve a full set of interpretations and processing history with or without probabilistic information to be passed along to later stages. We once again assume that XML markup is an appropriate format for expressing knowledge that can be created by a variety of external processes and exploited by a number of technologies in addition to TTS.

Since today's TTS systems typically cannot expect that their input be independently marked up for text normalization, they incorporate internal technology to perform this function. Future systems may piggyback on full natural language processing solutions developed for independent purposes. Presently, many incorporate minimal, TTS-specific hand-written rules [1], while others are loose agglomerations of modular, task-specific statistical evaluators [3].

For some purposes, an architecture that allows for a set or lattice of possible alternative expansions may be preferable to deterministic text normalization, like the *n-best* lists or *word graph* offered by the speech recognizers described in Chapter 13. Alternatives known to the system can be listed and ranked by probabilities that may be learnable from data. Later stages of processing (linguistic analysis or speech synthesis) can either add knowledge to the lattice structure or recover the best alternative, if needed. Consider the fragment "*at 8 am I . . .* " in some informal writing such as email. Given the flexibility of writing conventions for pronunciation, *am* could be realized as either *A. M.* (the numeric context seems to cue at times) or the auxiliary verb *am*. Both alternatives could be noted in a descriptive lattice of covering interpretations, with confidence measures if known (Table 14.2).

Table 14.2 Two alternative interpretations for sentence fragment "*At 8 am I ...*".

At 8 am I ...	At <time> eight am </time> I ...
At 8 am I ...	At <number> eight </number> am I ...

[5] SNOR, or Standard Normalized Orthographic Representation, is a uniform way of writing words and sentences that corresponds to spoken rendition. SNOR-format sentence texts are required as reference material for many Defense Advanced Research Project Agency and National Institutes of Standards and Technology-sponsored standard speech technology evaluation procedures.

If the potential ambiguity in the interpretation of *am* in the above pair of examples is simply noted, and the alternatives retained rather than suppressed, the choice can be made by a later stage of syntactic/semantic processing. Note another feature of this example—the rough irregular abbreviation form for antemeridian, which by prescriptive convention hopes that high-quality TTS processing can rely entirely on *standard* stylistic conventions. That observation also applies to the *obligatory* use of "?" for all questions.

Specific architectures for the text normalization component of TTS may be highly variable, depending on the system architect's answers to the following questions:

- Are cross-functional language processing resources mandated, or available?
- If so, are phonetic forms, with stress or accent, and normalized orthography, available?
- Is a full syntactic and semantic analysis of input text mandated, or available?
- Can the presenting application add interpretive knowledge to structure the input (text)?
- Are there interface or pipelining requirements that preclude lattice alternatives at every stage?

Because of this variability in requirements and resources, we do not attempt to formally specify a single, all-purpose architectural solution here. Rather, we concentrate on describing the text normalization challenges any system has to face. We note where solutions to these challenges are more readily realized under particular architectural assumptions.

All text normalization consists of two phases: identification of type, and expansion to SNOR or other unambiguous representation. Much of the identification phase, dealing with phenomena of sentence boundary determination, abbreviation expansion, number spell-out, etc., can be modeled as regular expression (see Chapter 11). This raises an interesting architectural issue. Imagine a system based entirely on regular *finite state transducers* (FST, see Chapter 11), as in [27], which enforces an appealing uniformity of processing mechanism and internal structure description. The FST permits a lattice-style representation that does not require premature resolution of any structural choice. An entire text analysis system can be based on such a representation. However, as long as a system confines its attention to issues that commonly come under the heading of text normalization, such as number formats, abbreviations, and sentence breaking, a simpler regular-expression-based uniform mechanism for rule specification and structure representation may be adequate.

Alternatively, TTS systems could make use of advanced tools such as, for example, the lex and yacc tools [17], which provide frameworks for writing customized lexical analyzers and context-free grammar parsers, respectively. In the discussion of typical text normalization requirements below, examples will be provided and then a fragment of Perl pattern-matching code will be shown that allows matching of the examples given. Perl notation [36] is used as a convenient short-hand representing any equivalent regular expressionparsing system and can be regarded as a subset of the functionality provided by any regular expression, FST, or context-free grammar tool set that a TTS software architect may choose to employ. Only a small subset of the simple, fairly standard Perl conventions

to employ. Only a small subset of the simple, fairly standard Perl conventions for regular expression matching are used, and comments are provided in our discussion of text normalization.

A text normalization system typically adds identification information to assist subsequent stages in their tasks. For example, if the TN subsystem has determined with some confidence that a given digit string is a phone number, it can associate XML-like tags with its output, identifying the corresponding normalized orthographic chunk as a candidate for special phone-number intonation. In addition, the identification tags can guide the lexical disambiguation of terms for other processes, like phonetic analysis in TTS systems and training data preparation for speech recognition.

Table 14.3 shows some examples of input fragments with a relaxed form of output normalized orthography. It illustrates a possible ambiguity in TN output. In the (contrived) example, the ambiguity is between a place name and a hypothetical individual named perhaps *Steve* or *Samuel* Asia. Two questions arise in such cases. The first is format of specification. The data between submodules in a TTS system can be passed (or be placed in a centrally viewable *blackboard* location) as tagged text or in a binary format. This is an implementation detail. Most important is that all possibilities known to the TN system be specified in the output, and that confidence measures from the TN, if any, be represented. For example, in many contexts, *South Asia* is the more likely spell-out of *S. Asia*, and this should be indicated implicitly by ordering output strings, or explicitly with probability numbers. The decision could then be delayed until one has enough information in the later module (like linguistic analysis) to make the decision in an informed manner.

Table 14.3 Examples of the normalized output using XML-like tags for text normalization.

Dr. King	<title> DOCTOR </title> KING
7%	<number>SEVEN<ratio>PERCENT</ratio> </number>
S. Asia	<toponym> SOUTH ASIA </toponym> OR <psn_name><initial>S</initial>ASIA</psn_name>

14.4.1. Abbreviations and Acronyms

As noted above, a period is an important but not completely reliable clue to the presence of an abbreviation. Periods may be omitted or misplaced in text for a variety of reasons. For similar reasons of stylistic variability and a writer's (lack of) care and skill, capitalization, another potentially important clue, can be variable as well. For example, all the representations of the abbreviation for *post script* listed below have been observed in actual mail and email. A system must therefore combine knowledge from a variety of contextual sources, such as document structure and origin, when resolving abbreviations:

PS. Don't forget your hat.
Ps. Don't forget your hat.
P.S. Don't forget your hat.
P.s. Don't forget your hat.

And *P.S.*, when examined out of context, could be personal name initials as well. Of course, a given TTS system's user may be satisfied with the simple spoken output */p iy ae s/* in cases such as the above, obviating the need for full interpretation. But at a minimum, when *fallback to letter pronunciation* is chosen, the TTS system must attempt to ensure that some obvious spell-out is not being overlooked. For example, a system should not render the title in *Dr. Jones* as letter names */d iy aa r/*.

Actually, any abbreviation is potentially ambiguous, and there are several distinct types of ambiguity. For example, there are abbreviations, typically quantity and measure terms, which can be realized in English as either plural or singular depending on their numeric coefficient, such as *mm* for *millimeter(s)*. This type of ambiguity can get especially tricky in the context of conventionally frozen items. For example, *9mm ammunition* is typically spoken as *nine millimeter ammunition* rather than *nine millimeters ammunition*.

Next, there are forms that can, with appropriate syntactic context, be interpreted either as abbreviations or as simple English words, such as in (inches), particularly at the end of sentences.

Finally, many, perhaps most, abbreviations have entirely different abbreviation spell-outs depending on semantic context, such as *DC* for *direct current* or *District of Columbia*. This variability makes it unlikely that any system ever performs perfectly. However, with sufficient training data, some statistical guidelines for interpretation of common abbreviations in context can be derived. Table 14.4 shows a few more examples of this most difficult type of ambiguity.

An advanced TTS system should attempt to convert reliably at least the following abbreviations:

- Title—Dr., MD, Mr., Mrs., Ms., St. (Saint), ... etc.
- Measure—ft., in., mm, cm (centimeter), kg (kilogram), ... etc.
- Place names—CO, LA, CA, DC, USA, St. (street), Dr. (drive), ... etc.

Table 14.4 Some ambiguous abbreviations.

CO	Colorado	commanding officer
	conscientious objector	carbon monoxide
IRA	Individual Retirement Account	Irish Republican Army
MD	Maryland	doctor of medicine
	muscular dystrophy	

Abbreviation disambiguation usually can be resolved by POS (part-of-speech) analysis. For example, whether *Dr.* is *Doctor* or *Drive* can be resolved by examining the POS features of the previous and following words. If the abbreviation is followed by a capitalized personal name, it can be expanded as *Doctor*, whereas if the abbreviation is preceded by a capitalized place name, a number, or an alphanumeric (like 120th), it will be expanded as *Drive*. Although the example above is resolved via a series of heuristic rules, the disambiguation (POS analysis) can also be done by a statistical approach. In [6], the POS tags are determined based on the most likely POS sequence using POS trigram and lexical-POS unigram. Since an abbreviation can often be distinguished by its POS feature, the most likely POS sequence of the sentence discovered by the trigram search then provides the best guess of the POS (thus the usage) for abbreviations. We describe POS tagging in more detail in Section 14.5.

Other than POS information, the lexical entries for abbreviations should include all features and alternatives necessary to generate a lattice of possible analyses. For example, a typical abbreviation's entry might include information as to whether it could be a word (like *in*), whether period(s) are optional or required, whether plural variants must be generated and if so under what circumstances, whether numerical specification is expected or required, etc.

Acronyms are words created from the first letters or parts of other words. For example, SCUBA is an acronym for *self-contained underwater breathing apparatus*. Generally, to qualify as a true acronym, a letter sequence should reflect normal language phonotactics, such as a reasonable alternation of consonants and vowels. From a TTS system's point of view, the distinctions between acronyms, abbreviations, and plain new or unknown words can be unclear. Many acronyms can be entered into the TTS system lexicon just as ordinary words would be. However, unknown acronyms (not listed in the lexicon) may occasionally be encountered. Although an acronym's case property can be a significant clue to identification, it is often unclear how to speak a given sequence of upper-case letters. Most TTS systems, failing to locate the sequence in the acronym dictionary, spell it out letter-by-letter. Other systems attempt to determine whether the sequence is inherently *speakable*. For example, DEC might be inherently speakable, while FCC is not formed according to normal word phonotactics. When something speakable is found, it is processed via the normal letter-to-sound rules, while something *unspeakable* would be spelled out letter-by-letter. Yet other systems might simply feed the sequence directly to the letter-to-sound rules (see Section 14.8), just as they would any other unknown word. As with all such problems, a larger lexicon usually provides superior results.

The general algorithm for abbreviations and acronyms expansion in text normalization is summarized in Algorithm 14.2. The algorithm assumes that tokenization and POS tagging have been done for the whole sentence. Abbreviation expansion is determined by the POS tags of the potential abbreviation candidates. Acronym expansion is done exclusively by table lookup, and letter-by-letter spell-out is used when acronyms cannot be found in the acronym table.

ALGORITHM 14.2: *ABBREVIATIONS AND ACRONYMS EXPANSION*

1. If word token *w* is not in abbreviation table and *w* contains only capital letters **goto 3.**
2. **Abbreviation Expansion**
   If the POS tag of *w* and the correspondent abbreviation match
   Abbreviation expansion by inserting SNOR and interpretive annotation tags
   Advance one word and **goto 1.**
3. **Acronym Expansion**
   If *w* is in the predefined acronym table
   Acronym expansion by inserting SNOR and interpretive annotation tags
   according to acronym expansion table
   else spell out *w* letter-by-letter
4. Advance one word and **goto 1.**

14.4.2. Number Formats

Numbers occur in a wide variety of formats and have a wide variety of contextually dependent reading styles. For example, the digits 370 in the context of the product name *IBM 370 mainframe computer* typically are read as *three seventy*, while in other contexts 370 would be read as *three hundred seventy* or *three hundred and seventy*. In a phone number, such as 370-1111, the string would normally be read as *three seven oh*, while in still other contexts it might be rendered as *three seven zero*. A text analysis system can incorporate rules, perhaps augmented by probabilities, for these situations, but might never achieve perfection in all cases. Phone numbers are a practical place to start, and their treatment illustrates some of the general issues relevant to the other number formats which are covered below.

14.4.2.1. Phone Numbers

Phone numbers may include prefixes and area codes and may have dashes and parentheses as separators. Examples are shown in Table 14.5.

The first two examples have *prefix* codes, while the next four have area codes with minor formatting differences. The final two examples are possible international-format phone numbers. A basic Perl regular expression pattern to subsume the commonality in all the local domestic numbers can be defined as follows:

```
$us_basic = '([0-9]{3}\-[0-9]{4})';
```

This defines a pattern subpart to match 3 digits, followed by a separator dash, followed by another 4 digits. Then the pattern to match the prefix type would be:

```
/([0-9]{1})[\/ -]($us_basic)/
```

Table 14.5 Some different written representations of phone numbers.

9-999-4118
9 345-5555
(617) 932-9209
(617) 932-9209
716-123-4568
409/845-2274
+49 (228) 550-381
+49-228-550-381

In the first example above, this leaves the system pattern variable $1 (corresponding to the first set of capture parentheses in the pattern) set to 9, and $2 (the second set of capture parentheses) set to 999-4118. Then a separate set of tables, indexed by the rule name and the pattern variable contents, could provide orthographic spell-outs for the digits. Clearly a balance has to be struck between the number of pattern variables provided in the expression and the overall complexity of the expression, vis-à-vis the complexity and sophistication of the indexing scheme of the spell-out tables. For example, the $us_basic could be defined to incorporate parentheses capture on the first three digits and the remaining four separately, which might lead to a simpler spell-out table in some cases.

The pattern to match the area code types could be:

```
/(\([0-9]{3}\))[\/ -]($us_basic)/
```

These patterns could be endlessly refined, expanded, and layered to match strings of almost arbitrary complexity. A balance has to be struck between number and complexity of distinct patterns. In any case, no matter how sophisticated the matching mechanism, arbitrary or at best probabilistic decisions have to be made in constructing a TTS system. For example, in matching an area code type, the rule architect must decide how much and what kind of whitespace separation the matching system tolerates between the area code and the rest of the number before a phone-number match is considered unlikely. Or, as another example, does the rule architect allow new lines or other formatting characters to appear between the area code and the basic phone number? These kinds of decisions must be explicitly considered, or made by default, and should be specified to a reasonable degree in user documentation. There are a great many other phone number formats and issues that are beyond the scope of this treatment.

Once a certain type of pattern requires a conversion to normalized orthography, the question of how to perform the conversion arises. The conversion characters can be aligned with the identification, so that conversion occurs implicitly during the pattern matching process. Another way is to separate the conversion from the identification phase. This may or may not lead to gains in efficiency and elimination of redundancy, depending on the

overall architecture of the system and whether and how components are expected to be reused. A version of this second approach is sketched here.

Suppose that the pattern match variable $1 has been set to 617 by one of the identification-phase pattern matches described above. Another list can provide pointers to conversion tables, indexed by the rule name or number and the variable name. So for the rule that can match area codes, the relevant entry would be:

```
Identification rule     Variable      Spellout table
Area-Phone              $1            LITERAL_DIGIT
```

The LITERAL_DIGIT spell-out rule set, when presented with the 617 character sequence (the value of $1), simply generates the normalized orthography *six one seven*, by table lookup. In this simple and straightforward approach, spell-out tables such as LITERAL_DIGIT can be reused for portions of a wide variety of identification rules. Other simple numeric spell-out tables would cover different styles of numeric reading, such as *pairwise* style (e.g., *six seventeen*), full decimal with tens, hundreds, thousands units (*six hundred seventeen*), and so on. Some spellout tables may require processing code to supplement the basic table lookup. Additional examples of spell-out tables are not provided for the various other types of text normalization entities exemplified below, but would function similarly.

14.4.2.2. Dates

Dates may be specified in a wide variety of formats, sometimes with a mixture of orthographic and numeric forms. Note that dates in TTS suffer from a mild form of the century-date-change uncertainty (the infamous Y2K bug), so a form such as 5/7/37 may in the future be ambiguous, in its full form, between 1937 and 2037. The safest course is to say as little as possible, i.e., "*five seven thirty seven*", or even "*May seventh, thirty seven*", rather than attempt "*May seventh, nineteen thirty seven*". Table 14.6 shows a variety of date formats and associated normalized orthography.

Table 14.6 Various date formats.

12/19/94 (US)	December nineteenth ninety four
19/12/94 (European)	December nineteenth ninety four
04/27/1992	April twenty seventh nineteen ninety two
May 27, 1995	May twenty seventh nineteen ninety five
July 4, 94	July fourth ninety four
1,994	one thousand nine hundred and ninety four
1994	nineteen ninety four

One issue that comes up with certain number formats, including dates, is range checking. A form like 13/19/94 is basically uninterpretable as a date. This kind of checking, if included in the initial pattern matching, may be slow and may increase formal requirements for power of the pattern matching system. Therefore, range checking can be done at spell-out time (see below) during normalized orthography generation, as long as a backtracking or redo option is present. If range checking is desired as part of the basic identification phase of text normalization, some regular expression matching systems allow for extensions. For example, the following pattern variable matches only numbers less than or equal to 12, the valid month specifications. It can be included as part of a larger, more complex date matching *pattern:*

```
$month = '/(0[123456789]/1[012]/'
```

14.4.2.3. Times

Times may include hours, minute, seconds, and duration specifications as shown in Table 14.7. Time formats exemplify yet another area where linguistic concerns have to intersect with architecture. If simple, flat normalized orthography is generated during a text normalization phase, a later stage may still find a form like *am* ambiguous in pronunciation. If a lattice of alternative interpretations is provided, it should be supplemented with interpretive information on the linguistic status of the alternative text analyses. Alternatively, a single best guess can be made, but even in this case, some kind of interpretive information indicating the status of the choice as, e.g., a time expression, should be provided for later stages of syntactic, semantic, and prosodic interpretation. This reiterates the importance of TTS text analysis systems to generate interpretive annotations tags for subsequent modules' use whenever possible, as discussed in Section 14.4. In some cases, unique text formatting of the choice, corresponding to the system's lexical contents, may be sufficient. That is, in some systems, generation of *A.M.*, for example, may uniquely correspond to the lexicon's entry for that portion of a time expression, which specifies the desired pronunciation and grammatical treatment.

Table 14.7 Several examples for time expressions.

11:15	eleven fifteen
8:30 pm	eight thirty pm
5:20 am	five twenty am
12:15:20	twelve hours fifteen minutes and twenty seconds
07:55:46	seven hours fifty-five minutes and forty-six seconds

14.4.2.4. Money and Currency

As illustrated in Table 14.8, money and currency processing should correctly handle at least the currency indications $, £, DM, ¥, and €, standing for dollars, British pounds, deutsche marks, Japanese yen, and euros, respectively. In general, $ and £ have to precede the numeral; DM, ¥, and € have to follow the numeral. Other currencies are often written in full words and have to follow the numeral, though abbreviations for these are sometimes found, such as *100 francs* and *20 lira.*

Table 14.8 Several money and currency expressions.

$40	forty dollars
£200	two hundred pounds
5 ¥	five yen
25 DM	twenty five deutsche marks
300 €	three hundred euros

14.4.2.5. Account Numbers

Account numbers may refer to bank accounts or social security numbers. Commercial product part numbers often have these kinds of formats as well. In some cases these cannot be readily distinguished from mathematical expressions or even phone numbers. Some examples are shown below:

```
123456-987-125456
000-1254887-87
049-85-5489
```

The other popular number format is that of credit card number, such as

```
4446-2289-2465-7065
3745-122267-22465
```

To process formats like these, it may eventually be desirable for TTS systems to provide customization capabilities analogous to the pronunciation customization features for words found in current TTS systems. Regular expression formalisms of the type exemplified above for phone number, would, if exposed to applications and developers through suitable editors, be adequate for most such needs.

14.4.2.6. Ordinal Numbers

Ordinal numbers are those referring to rank or placement in a series. Examples include:

1st, 2nd, 3rd, 4th, 10th, 11th, 12th, 20th, 100th, 1000th, etc.
1st, 2nd, 3rd, 4th, 10th, 11th, 12th, 20th, 21st, 32nd, 100th, 1000th, etc.

The system's ordinal processing may also be used to generate the denominators of fractions, except for halves, as shown in Table 14.9. Notice that the ordinal must be plural for numerators other than 1.

Table 14.9 Some examples of fractions.

1/2	one half
1/3	one third
1/4	one quarter or one fourth
1/10	one tenth
3/10	three tenths

14.4.2.7. Cardinal Numbers

Cardinal numbers are, loosely speaking, those forms used in simple counting or the statement of amounts. If a given sequence of digits fails to fit any of the more complex formats above, it may be a simple cardinal number. These may be explicitly negative or positive or assumed positive. They may include decimal or fractional specifications. They may be read in several different styles, depending on context and/or aesthetic preferences. Table 14.10 gives some examples of cardinal numbers and alternatives for normalized orthography.

The number-expansion algorithm is summarized in Algorithm 14.3. In this algorithm the text normalization module maintains an extensive pattern table. Each pattern in the table contains its associated pattern in regular expression or Perl format along with a pointer to a rule in the conversion table, which guides the expansion process.

Table 14.10 Some cardinal number types.

123	one two three	one hundred (and) twenty three
1,230	one thousand two hundred (and) thirty	
2426	two four two six	twenty four twenty six
	two thousand four hundred (and) twenty six	

A regular expression to match well-formed cardinals with commas grouping chunks of three digits of the type from 1,000,000 to 999,999,999 might appear as:

```
if ($item =~ /^([0-9]{1,3}),([0-9]{3}),([0-9]{3})/
 {      $NewFrame->{"millions"} = $1;
   $NewFrame->{"thousands"} = $2;
   $NewFrame->{"hundreds"} = $3;
    print "Grouped cardinal found: $item\n";
   return $NewFrame;   }
```

ALGORITHM 14.3: *NUMBER EXPANSION*

1. **Pattern Matching**
   If a match is found **goto 2.**
   else **goto 3.**
2. **Number Expansion**
   Insert SNOR and interpretive annotation tags according to the associated rule
   Advance the pointer to the right of the match pattern and **goto 1.**
3. **Finish**

14.4.3. Domain-Specific Tags

In keeping with the theme of this section—that is, the increasing importance of independently generated precise markup of text entities—we present a little-used but interesting example.

14.4.3.1. Mathematical Expressions

Mathematical expressions are regarded by some systems as the domain of special-purpose processors. It is a serious question how far to go in mathematical expression parsing, since providing some capability in this area may raise users' expectations to an unrealistic level. The World Wide Web Consortium has developed MathML (mathematical markup language) [34], which provides a standard way of describing math expressions. MathML is an XML extension for describing mathematical expression structure and content to enable mathematics to be served, received, and processed on the Web, similar to the function HTML has performed for text. As XML becomes increasingly pervasive, MathML could possibly be used to guide interpretation of mathematical expressions. For the notation $(x + 2)^2$ a possible MathML representation such as that below might serve as an initial guide for a spoken rendition.

```
<EXPR>
 <EXPR>
 x
 <PLUS/>
 2
 </EXPR>
 <POWER/>
 2
</EXPR>
```

This might be generated by an application or by a specialized preprocessor within the TTS system itself. Prosodic rules or data tables appropriate for math expressions could then be triggered.

14.4.3.2. Chemical Formulae

As XML becomes increasingly common and exploitable by TTS text normalization, other areas follow. For example, Chemical Markup Language (CML [22]) now provides a standard way to describe molecular structure or chemical formulae. CML is an example of how standard conventions for text markup are expected increasingly to replace ad hoc, TTS-internal heuristics.

In CML, the chemical formula C_2OCOH_4 would appear as:

```
<FORMULA>
   <XVAR BUILTIN="STOICH">
   C C O C O H H H H
   </XVAR>
</FORMULA>
```

It seems reasonable to expect that TTS engines of the future will be increasingly devoted to interpreting such precise conventions in high-quality speech renditions rather than endlessly replicating NL heuristics that fail as often as they succeed in guessing the identity of raw text strings.

14.4.4. Miscellaneous Formats

A random list illustrating the range of other types of phenomena for which an English-oriented TTS text analysis module must generate normalized orthography might include:

- Approximately/tilde: The symbol ~ is spoken as *approximately* before (Arabic) numeral or currency amount, otherwise it is the character named *tilde*.

- Folding of accented Roman characters to *nearest* plain version: If the TTS system has no knowledge of dealing with foreign languages, like French or German, a table of folding characters can be provided so that for a term such as *Über-mensch*, rather than spell out the word *Über*, or ignore it, the system can convert it to its *nearest* English-orthography equivalent: *Uber*. The ultimate way to process such foreign words should integrate a language identification module with a multi-lingual TTS system, so that language-specific knowledge can be utilized to produce appropriate text normalization of all text.
- Rather than simply ignore high ASCII characters in English (characters from 128 to 255), the text analysis lexicon can incorporate a table that gives *character names* to all the printable high ASCII characters. These names are either the full Unicode character names, or an abbreviated form of the Unicode names. This would allow speaking the names of characters like © (*copyright sign*), ™ (*trademark*), @ (*at*), ® (*registered mark*), and so on.
- Asterisk: in email, the symbol '*' may be used for emphasis and for setting off an item for special attention. The text analysis module can introduce a little pause to indicate possible emphasis when this situation is detected. For the example of "*Larry has *never* been here*," this may be suppressed for asterisks spanning two or more words. In some texts, a word or phrase appearing completely in UPPER CASE may also be a signal for special emphasis.
- Emoticons: There are several possible emoticons (emotion icons).

  1. :-) or :) SMILEY FACE (humor, laughter, friendliness, sarcasm)
  2. :-(or :(FROWNING FACE (sadness, anger, or disapproval)
  3. ;-) or ;) WINKING SMILEY FACE (naughty)
  4. :-D OPEN-MOUTHED SMILEY FACE (laughing out loud)

  Smileys, of which there are dozens of types, may be tacked onto word start or word end or even occur interword without spaces, as in the following examples.

  :)hi!
  Hi:)
  Hi:)Hi!

14.5. LINGUISTIC ANALYSIS

Linguistic analysis (sometimes also referred to as syntactic and semantic parsing) of natural language (NL) constitutes a major independent research field. Often commercial TTS systems incorporate some minimal parsing heuristics developed strictly for TTS. Alternatively, the TTS systems can also take advantage of independently motivated natural language proc-

essing (NLP) systems, which can produce structural and semantic information about sentences. Such linguistically analyzed documents can be used for many purposes other than TTS—information retrieval, machine translation system training, etc.

Provision of some parsing capability is useful to TTS systems in several areas. Parsers may be used in disambiguating the text normalization alternatives described above. Additionally, syntactic/semantic analysis can help to resolve grammatical features of individual words that may vary in pronunciation according to sense or abstract inflection, such as *read*. Finally, parsing can lay a foundation for derivation of a prosodic structure useful in determining segmental duration and pitch contour.

The fundamental types of information desired for TTS from a parsing analysis are summarized below:

- Word part of speech (POS) or word type, e.g., proper name or verb.
- Word sense, e.g., river *bank* vs. money *bank*.
- Phrasal cohesion of words, such as idioms, syntactic phrases, clauses, sentences.
- Modification relations among words.
- Anaphora (co-reference) and synonymy among words and phrases.
- Syntactic type identification, such as questions, quotes, commands, etc.
- Semantic focus identification (emphasis).
- Semantic type and speech act identification, such as requesting, informing, narrating, etc.
- Genre and style analysis.

Here we confine ourselves to discussion of the kind of information that a good parser could, in principle, provide to enable the TTS-specific functionality.

Linguistic analysis supports the phonetic analysis and prosodic generation phases. The modules of phonetic analysis are covered in Sections 14.6, 14.7, and 14.8. A linguistic parser can contribute in several ways to the process of generating (symbolic) phonetic forms from orthographic words found in text. One function of a parser is to provide accurate part-of-speech (POS) labels. This can aid in resolving the pronunciation of several hundred American English homographs, such as *object* and *absent*. Homographs are discussed in greater detail in Section 14.6. Parsers can also aid in identifying names and other special classes of vocabulary for which specialized pronunciation rule sets may exist [32].

Prosody generation deals mainly with assignment of segmental duration and pitch contour that have close relationship with prosodic phrasing (pause placement) and accentuation. Parsing can contribute useful information, such as the syntactic type of an utterance. (e.g., yes/no question contours typically differ from *wh*-question contours, though both are marked simply by '?' in text), as well as semantic relations of synonymy, anaphora, and focus that may affect accentuation and prosodic phrasing. Information from discourse analysis and text genre characterization may affect pitch range and voice quality settings. Further

examination of the contribution of parsing specifically to prosodic phrasing, accentuation, and other prosodic interpretation is provided in Chapter 15.

As mentioned earlier, TTS can employ either a general-purpose NL analysis engine or a pipeline of a number of very narrowly targeted, special-purpose NL modules together for the requirement of TTS linguistic analysis. Although we focus on linguistic information for supporting phonetic analysis and prosody generation here, a lot of the information and services are beneficial to document structure detection and text normalization described in previous sections.

The minimum requirement for such a linguistic analysis module is to include a lexicon of the *closed-class* function words, of which only several hundred exist in English (at most), and perhaps homographs. In addition, a minimal set of modular functions or services would include:

- *Sentence breaking*—Sentence breaking has been discussed in Section 14.3.4 above.
- *POS tagging*—POS tagging can be regarded as a two-stage process. The first is POS guessing, which is the process of determining, through a combination of a (possibly small) dictionary and some morphological heuristics or a specialized morphological parser, the POS categories that might be appropriate for a given input term in isolation. The second is POS choosing—that is, the resolution of the POS in context, via local short-window syntactic rules, perhaps combined with probabilistic distribution for the POS guesses of a given word. Sometimes the guessing and choosing functions are combined in a single statistical framework. In [6], lexical probabilities are unigram frequencies of assignments of categories to words estimated from corpora. In the original formulation of the model, the lexical probabilities [$P(c_i \mid w_i)$, where c_i is the hypothesized POS for word w_i], were estimated from the hand-tagged Brown corpus [8]. For Example, the word *see* appeared 771 times as a verb and once as an interjection. Thus the probability that *see* is a verb is estimated to be 771/772 or 0.99. Trigrams are used for contextual probability [$P(c_i \mid c_{i-1}c_{i-2}\cdots c_1) = P(c_i \mid c_{i-1}c_{i-2})$]. Lexical probabilities and trigrams over category sequences are used to score all possible assignments of categories to words for a given input word sequence. The entire set of possible assignments of categories to words in sequence is calculated, and the best-scoring sequence is used. Likewise, simple methods have been used to detect noun phrases (NPs), which can be useful in assigning pronunciation, stress, and prosody. The method described in [6] relies on a table of probabilities for inserting an NP begin bracket '[' between any two POS categories, and similarly for an NP end bracket ']'. This was also trained on the POS-labeled Brown corpus, with further augmentation for the NP labels. For example, the probability of inserting an NP begin bracket after an article was found to be

much lower than that of begin-bracket insertion between a verb and a noun, thus automatically replicating human intuition.

- *Homograph disambiguation*—Homograph disambiguation in general refers to the case of words with the same orthographic representation (written form) but having different semantic meanings and sometimes even different pronunciations. Sometimes it is also referred as sense disambiguation. Examples include "The boy used the *bat* to hit a home run" vs. "We saw a large *bat* in the zoo" (the pronunciation is the same for two *bat*) and "You *record* your voice" vs. "I'd like to buy that *record*" (the pronunciations are different for the two *record*). The linguistic analysis module should at least try to resolve the ambiguity for the case of different pronunciations because it is absolutely required for correct phonetic rendering. Typically, the ambiguity can be resolved based on POS and lexical features. Homograph disambiguation is described in detail in Section 14.6.
- *Noun phrase (NP) and clause detection*—Basic NP and clause information could be critical for a prosodic generation module to generate segmental durations. It also provides useful cues to introduce necessary pauses for intelligibility and naturalness. Phrase and clause structure are well covered in any parsing techniques.
- *Sentence type identification*—Sentence types (declarative, yes-no question, etc.) are critical for macro-level prosody for the sentence. Typical techniques for identifying sentence types have been covered in Section 14.3.4.

If a more sophisticated parser is available, a richer analysis can be derived. A so-called *shallow parse* is one that shows syntactic bracketing and phrase type, based on the POS of words contained in the phrases. A training corpus of shallow-parsed sentences has been created for the Linguistic Data Consortium [16]. The following example illustrates a shallow parse for sentence : "For six years, Marshall Hahn Jr. has made corporate acquisitions in the George Bush mode: kind and gentle."

```
For/IN[six/CD years/NNS],/,[T./NNP Marshall/NNP
Hahn/NNP Jr./NNP]has/VBZ made/VBN[corporate/JJ acquisi-
tions/NNS]in/IN[the/DT George/NNP Bush/NNP mode/NN]
:/:[kind/JJ]and/CC[gentle/JJ]./.
```

The POS labels used in this example are described in Chapter 2 (Table 2.14). A TTS system uses the POS labels in the parse to decide alternative pronunciations and to assign differing degrees of prosodic prominence. Additionally, the bracketing might assist in deciding where to place pauses for great intelligibility. A fuller parse would incorporate more higher-order structure, including sentence type identification, and more semantic analysis, including co-reference.

14.6. HOMOGRAPH DISAMBIGUATION

For written languages, *sense* ambiguities occur when words have different syntactic/semantic meanings. Those words with different senses are called *polysemous* words. For example, *bat* could mean either a kind of animal or the equipment to hit a baseball. Since the pronunciations for the two different senses of *bat* are identical, we are in general only concerned[6] about the other type of polysemous words that are homographs (spelled alike but vary in pronunciation), such as *bass* for a kind of fish (*/b ae s/*) or an instrument (*/b ey s/*).

Homograph variation can often be resolved on POS (grammatical) category. Examples include *object*, *minute*, *bow*, *bass*, *absent*, etc. Unfortunately, correct determination of POS (whether by a parsing system or statistical methods) is not always sufficient to resolve pronunciation alternatives. For example, simply knowing that the form *bow* is a noun does not allow us to distinguish the pronunciation appropriate for the instrument of archery from that for the front part of a boat. Even more subtle is the pronunciation of *read* in "If you *read* the book, he'll be angry." Without contextual clues, even human readers cannot resolve the pronunciation of *read* from the given sentence alone. Even though the past tense is more likely in some sense, deep semantic and/or discourse analysis would be required to resolve the tense ambiguity.

Several hundred English homographs extracted from the 1974 *Oxford Advanced Learners Dictionary* are listed in [10]. Here are some examples:

- Stress homographs: noun with front-stress vowel, verb with end-stress vowel
  "an *absent* boy" vs. "Do you choose to *absent* yourself?"
- Voicing: noun/verb or adjective/verb distinction made by voice final consonant
  "They will *abuse* him." vs. "They won't take *abuse*."
- –ate words: noun/adjective sense uses schwa, verb sense uses a full vowel
  "He will *graduate*." vs. "He is a *graduate*."
- Double stress: front-stressed before noun, end-stressed when final in phrase
  "an *overnight* bag" vs. "Are you staying *overnight*?"
- -ed adjectives with matching verb past tenses
  "He is a *learned* man." vs. "He *learned* to play piano."
- Ambiguous abbreviations: already described in Section 14.4.1
  in, am, SAT (Saturday vs. Standard Aptitude Test)
- Borrowed words from other languages—They could sometimes be distinguishable based on capitalization.
  "El Camino *Real* road in California" vs. "*real* world"
  "*polish* shoes" vs. "*Polish* accent"

[6] Sometimes, a polysemous word with the same pronunciation could have impact for prosodic generation because different semantic properties could have different accentuation effects. Therefore, a high-quality TTS system can definitely be benefited from word-sense disambiguation beyond homograph disambiguation.

- Miscellaneous
  "The *sewer* overflowed." vs. "a *sewer* is not a tailor."
  "He *moped* since his parents refused to buy a *moped*."
  "*Agape* is a Greek word." vs. "His mouth was *agape*."

As discussed earlier, abbreviation/acronym expansion and linguistic analysis described in Sections 14.4.1 and 14.5 are two main sources of information for TTS systems to resolve homograph ambiguities.

We close this section by introducing two special sources of pronunciation ambiguity that are not fully addressed by current TTS systems. The first one is a variation of dialects (or even personal dialect—*idiolect*). For example, some might say *tom*[*ey*]*to*, while some others might say *tom*[*aa*]*to*. Another example is that Boston natives tend to reduce the */r/* sound in sentences like "Park your car in Harvard yard." Similarly, some people use the spelling pronunciation *in-ter-es-ting* as opposed to *intristing*. Finally, speech rate and formality level can influence pronunciation. For example, the */g/* sound in *recognize* may be omitted in faster speech. It might be a sensible decision to output all possible pronunciations as a multiple pronunciation list and hope the synthesis back end picks the one with better acoustic/prosodic voice rendition. While true homographs may be resolved by linguistic and discourse analysis, achieving a consistent presentation of dialectal and stylistic variation is an even more difficult research challenge.

The other special source of ambiguity in TTS is somewhat different from what we have considered so far, but may be a concern in some markets. Most borrowed or foreign single words and place names are realized naturally with pronunciation normalized to the main presentation language. Going beyond that, language detection refers to the ability of a TTS system to recognize the intended language of a multiword stretch of text. For example, consider the fragment "Well, as for the next department head, that is simply *une chose entendue*." The French phrase "*une chose entendue*" (something clearly understood) might be realized in a proper French accent and phone pronunciation by a bilingual English/French reader. For a TTS system to mimic the best performance, the system must have:

- language identification capability
- dictionaries and rules for both languages
- voice rendition capability for both languages

14.7. MORPHOLOGICAL ANALYSIS

General issues in morphology are covered in Chapter 2. Here, we consider issues of relating a surface orthographic form to its pronunciation by analyzing its component morphemes, which are minimal, meaningful elements of words, such as prefixes, suffixes, and stem words themselves. This decomposition process is referred as *morphological analysis* [28]. When a dictionary does not list a given orthographic form explicitly, it is sometimes possible to analyze the new word in terms of shorter forms already present. These shorter forms

may combine as prefixes, one or more stems or roots, and suffixes to generate new forms. If a word can be so analyzed, the listed pronunciations of the pieces can be combined, perhaps with some adjustment (phonological rules), to yield a phonetic form for the word as a whole.

The prefixes and suffixes are generally considered bound, in the sense that they cannot stand alone but must combine with a stem. A stem, however, can stand alone. A word such as *establishment* may be decomposed into a "stem" *establish* and a suffix *-ment*. In practice, it is not always clear where this kind of analysis should stop. That is, should a system attempt to further decompose the stem *establish* into *establ* and *-ish*? These kinds of questions ultimately belong to etymology, the study of word origins, and there is no final answer. However, for practical purposes, having three classes of entries corresponding to prefixes, stems, and suffixes, where the uses of the affixes are intuitively obvious to educated native speakers, is usually sufficient. In practical language engineering, a difference that makes no difference is no difference, and unless there is a substantial gain in compression or analytical power, it is best to be conservative and list only obvious and highly productive affixes.

The English language presents numerous genuine puzzles in morphological analysis. For example, there is the issue of abstraction: is the word *geese* one morpheme, or two (base *goose* + abstract pluralizing morpheme)? For practical TTS systems, relying on large dictionaries, it is generally best to deal with concrete, observable forms where possible. In such a lexically oriented system, the word geese probably should appear in the lexicon as such, with attached grammatical features including plurality. Likewise, it is simpler to include *children* in the lexical listing rather than create a special pluralizing suffix *-ren* whose use is restricted to the single base *child*.

The morphological analyzer must attempt to cover an input word in terms of the affixes and stems listed in the morphological lexicon. The covering(s) proposed must be legal sequences of forms, so that often a word grammar is supplied to express the allowable patterns of combinations. A word grammar might, for example, restrict suffixation to the final or rightmost stem of a compound, thus allowing plurality on the final element of *businessmen* but not in the initial stem (*businessesman*). In support of the word grammar, all stems and affixes in the lexicon would be listed with morphological combinatory class specifications, usually subtyped in accordance with the base POS categories of the lexicon entries. That is, verbs would typically accept a different set of affixes than nouns or adjectives. In addition, spelling changes that sometimes accompany affixation must be recognized and undone during analysis. For example, the word *stopping* has undergone final consonant doubling as part of accommodating the suffix *ing*.

A morphological analysis system might be as simple as a set of *suffix-stripping* rules for English. If a word cannot be found in the lexicon, a suffix-stripping rule can be applied to first strip out the possible suffix, including *–s, -'s, -ing, -ed, -est, -ment*, etc. If the stripped form can be found in the lexicon, a morphological decomposition is attained. Similarly, *prefix-stripping* rules can be applied to find prefix-stem decomposition for prefixes like *in-, un-, non-, pre-, sub-, etc.*, although in general prefix stripping is less reliable.

Suffix and prefix stripping gives an analysis for many common inflected and some derived words such as *helped, cats, establishment, unsafe, predetermine, subword*, etc. It helps in saving system storage. However, it does not account for compounding, issues of legality

of sequence (word grammar), or spelling changes. It can also make mistakes (from a synchronic point of view: *basement* is not *base* + *-ment*), some of which will have consequences in TTS rendition. A more sophisticated version could be constructed by adding elements such as POS type on each suffix/prefix for a rudimentary legality check on combinations. However, a truly robust morphological capability would require more powerful formal machinery and a more thorough analysis. Therefore, adding irregular morphological formation into a system dictionary is always a desirable solution.

Finally, sometimes in commercial product names the compounding structure is signaled by word-medial case differences, e.g., AltaVista™, which can aid phonetic conversion algorithms. These can be treated as two separate words and will often sound more natural if rendered with two separate main stresses. This type of decomposition can be expanded to find compound words that are formed by two separate nouns. Standard morphological analysis algorithms employing suffix/prefix stripping and compound word decomposition are summarized in Algorithm 14.4. Note that the algorithm can be easily modified to handle words constructed by a combination of prefix, suffix, and compound.

ALGORITHM 14.4: *MORPHOLOGICAL ANALYSIS*

1. **Dictionary Lookup**
   Look up word w in lexicon
   If found
      Output attributes of the found lexical entry and exit
2. **Suffix Stripping**
   If word ends in *-s, -'s, -ing, -ed, -est, -ment,* etc.
      Strip the suffix from word w to form u
   If stripped form u found in lexicon
      Output attributes of the stem and suffix and exit
3. **Prefix Stripping**
   If word begins with *in-, un-, non-, pre-, sub-,* etc.
      Strip the prefix from word w to form u
   If stripped form u found in lexicon
      Output attributes of the prefix and stem and exit
4. **Compound word decomposition**
   If detect word-medial case differences within word w
      Break word w into a multiple words u_1, u_2, u_3, ...according to case changes
      For words u_1, u_2, u_3, **goto** 1.
   Else if word w can be decomposed into two nouns u_1, u_2 in lexicon
      Output attributes of the u_1, u_2 and exit
5. Pass word w to letter-to-sound module

14.8. LETTER-TO-SOUND CONVERSION

The best resource for generating (symbolic) phonetic forms from words is an extensive word list. The accuracy and efficiency of such a solution is limited only by the time, effort, and knowledge brought to bear on the dictionary construction process. As described in Section 14.2, a general lexicon service is a critical resource for the TTS system. Thus, the first and the most reliable way for grapheme-to-phoneme conversion is via dictionary lookup.

Where direct dictionary lookup fails, rules may be used to generate phonetic forms. Under earlier naïve assumptions about the regularity and coverage of simple descriptions of English orthography, rules have traditionally been viewed as the primary source of phonetic conversion knowledge, since no dictionary covers every input form and the TTS system must always be able to speak any word. A general *letter-to-sound* (LTS) conversion is thus required to provide phonetic pronunciation for any sequence of letters.

Inspired by the phonetic languages, letter-to-sound conversion is usually carried out by a set of rules. These rules can be thought of as dictionaries of fragments with some special conventions about lookup and context. Typically, rules for phonetic conversion have mimicked phonological rewriting in phonological theory [5], including conventions of ordering, such as *most specific first*. In phonological rules, a target is given and the rewrite is indicated, with context following. For example, a set of rules that changes orthographic *k* to a velar plosive */k/* except when the *k* is word-initial ('[') followed by *n* might appear as:

```
k -> /sil/ % [ _ n
k -> /k/
```

The rule above reads that *k* is rewritten as (phonetic) silence when in word initial position and followed by *n*, otherwise *k* is rewritten as (phonetic) */k/*. The underscore in the first line is a placeholder for the *k* itself in specifying the context. This little set properly treats *k* in *knight*, *darkness*, and *kitten*. These are formally powerful, context-sensitive rules. Generally a TTS system require hundreds or even thousands of such rules to cover words not appearing in the system dictionary or *exception list*. Typically rules are specified in terms of single-letter targets, such as the example for *k* above. However, some systems may have rules for longer fragments, such as the special vowel and consonant combinations in words like *neighbor* and *weigh*. In practice, a binary format for compression, a corresponding fragment matching capability, and a rule index must be defined for efficient system deployment.

Rules of this type are tedious to develop manually. As with any *expert system*, it is difficult to anticipate all possible relevant cases and sometimes hard to check for rule interference and redundancy. In any case, the rules must be verified over a test list of words with known transcriptions. Generally, if prediction of main stress location is not attempted, such rules might account for up to 70% of the words in a test corpus of general English. If prediction of main stress is attempted, the percentage of correct phonetic pronunciations is much lower, perhaps below 50%. The correct prediction of stress depends in part on morphology, which is not typically explicitly attempted in this type of simple rule system (though fragments corresponding to affixes are frequently used, such as *tion -> /ah ax n/*). Certainly, such rules can be made to approach dictionary accuracy, as longer and more explicit mor-

phological fragments are included. One extreme case is to create one specific rule (containing exact contexts for the whole word) for each word in the dictionary. Obviously this is not desirable, since it is equivalent to putting the word along with its phonetic pronunciation in the dictionary.

In view of how costly it is to develop LTS rules, particularly for a new language, attempts have been made recently to automate the acquisition of LTS conversion rules. These self-organizing methods believe that, given a set of words with correct phonetic transcriptions (the offline dictionary), an automated learning system could capture significant generalizations. Among them, classification and regression trees (CART) have been demonstrated to give satisfactory performances for letter-to-sound conversion. For basic and theoretic description of CART, please refer to Chapter 4.

In the system described in [14], CART methods and phoneme trigrams were used to construct an accurate conversion procedure. All of the experiments were carried on two databases. The first is the NETALK [25], which has hand-labeled alignment between letter and phoneme transcriptions. The second is the CMU dictionary, which does not have any alignment information. The NETALK database consists of 19,940 entries, of which 14,955 were randomly selected as a training set and the remaining 4951 were reserved for testing. Those 4951 words correspond to 4985 entries in the database because of multiple pronunciations. The hand-labeled alignments were used directly to train the CART for LTS conversion. The CMU dictionary has more than 100,000 words, of which the top 60,000 words were selected based on unigram frequencies trained from North American Business News. Among them, 52,415 were used for training and 9719 reserved for testing. Due to multiple pronunciations, those 9719 words have 10,520 entries in the dictionary. Due to lack of alignment information, dynamic programming was used to align each letter to the corresponding phoneme before training the LTS CART.

The basic CART component includes a set of *yes-no* questions and a procedure to select the best question at each node to grow the tree from the root. The basic *yes-no* question for LTS conversion looks like "*Is the second right letter 'p'?*" or "*Is the first left output phoneme /ay/?*" The questions for letters could be on either the left or the right side. For phones, only questions on the left side were used, for simplicity. The range of question positions must be long enough to cover the long-distance phonological variations. It was found that the 11-letter window (5 for left letter context and 5 for right letter context) and 3-phoneme window for left phoneme context are generally sufficient. A primitive set of questions would be the set of all the singleton questions about each letter or phoneme identity. When growing the tree, the question that had the best entropy reduction was chosen at each node. We observed that if we allow the node to have a complex question that is a combination of primitive questions, the depth of the tree will be greatly reduced and the performance improved. For example, the complex question "*Is the second left letter 't' and the first left letter 'i' and the first right letter 'n'?*" can capture '*o*' in common suffix "*tion*" and convert it to the right phoneme. Complex questions can also alleviate the data fragment problem caused by greedy nature of the CART algorithm. This way of finding such complex questions is similar to those used in Chapter 4. The baseline system built using the above techniques has error rates as listed in Table 14.11.

Table 14.11 LTS baseline results using CART [13].

Database	Phoneme	Word
CMU Lexicon	9.7%	35.0%
NETTALK	9.5%	42.3%

The CART LTS system [14] further improved the accuracy of the system via the following extensions and refinements:

- *Phoneme trigram rescoring*: A statistical model of phoneme co-occurrence, or phonotactics, was constructed over the training set. A phonemic trigram was generated from the training samples with back-off smoothing, and this was used to rescore the n-best list generated by LTS.
- *Multiple tree combination:* The training data was partitioned into two parts and two trees were trained. When the performance of these two trees was tested, it was found that they had a great overlap but also behaved differently, as each had a different focus region. Combining them together greatly improved the coverage. To get a better overall accuracy, the tree trained by all the samples was used together with two other trees, each trained by half of the samples. The leaf distributions of three trees were interpolated together with equal weights and then phonemic trigram was used to rescore the n-best output lists.

By incrementally experimenting with addition of these extensions and refinements, the results improved, as shown in Table 14.12.

These experiments did not include prediction of stress location. Stress prediction is a difficult problem, as we pointed out earlier. It requires information beyond the letter string. In principle, one can incorporate more lexical information, including POS and morphologic information, into the CART LTS framework, so it can be more powerful to learn the phonetic correspondence between the letter string and lexical properties.

Table 14.12 LTS using multiple trees and phonemic trigram rescoring [13].

Database	Phoneme	Word
CMU Lexicon	8.2%	26.9%
NETTALK	8.1%	34.2%

14.9. EVALUATION

Ever since the early days of TTS research [21, 31], evaluation has been considered an integral part of the development of TTS systems. End users and application developers are

mostly interested in the end-to-end evaluation of TTS systems. This *monolithic* type of whole-system evaluation is often referred to as *black-box* evaluation. On the other hand, *modular* (component) testing is more appropriate for TTS researchers when working with isolated components of the TTS system, for diagnosis or regression testing. We often refer to this type of evaluation as *glass-box* evaluation. We discuss the modular evaluations in each modular TTS chapter, while leaving the evaluation of the whole system to Chapter 16.

For text and phonetic analysis, automated, analytic, and objective evaluation is usually feasible, because the input and output of such module is relatively well defined. The evaluation focuses mainly on symbolic and linguistic level in contrast to the acoustic level, with which prosodic generation and speech synthesis modules need to deal. Such tests usually involve establishing a test corpus of correctly tagged examples of the tested materials, which can be automatically checked against the output of a text analysis module. It is not particularly productive to discuss such testing in the abstract, since the test features must closely track each system's design and implementation. Nevertheless, a few typical areas for testing can be noted. In general, tests are simultaneously testing the linguistic model and content as well as the software implementation of a system, so whenever a discrepancy arises, both possible sources of error must be considered.

For automatic detection of document structures, the evaluation typically focuses on sentence breaking and sentence type detection. Since the definitions of these two types of document structures are straightforward, a standard evaluation database can be easily established.

In the basic level, the evaluation for the text normalization component should include large regression test databases of text micro-entities: addresses, Internet and email entities, numbers in many formats (ordinal, cardinal, mathematical, phone, currency, etc.), titles, and abbreviations in a variety of contexts. These would be paired with the correct reference forms in something like the SNOR used in ASR output evaluation. In its simplest form, this would consist of a database of automatically checkable paired entries like 7% vs. *seven percent*, and $1.20 vs. *one dollar and twenty cents*. If you want to evaluate the semantic capability of text normalization, the regression database might include markups for semantic tags, so that we have 7% vs. "<number>SEVEN<ratio>PERCENT</ratio></number>", and $1.20 vs. "<money>ONE DOLLAR AND TWENTY CENTS</money>". The regression database could include domain-specific entries. This implies some dependence on the system's API—its markup capabilities or tag set. In the examples given in Table 14.13, the first one is a desirable output for domain-independent input, while the second one is suitable for normalization of the same expression in mathematical formula domain.

Table 14.13 Two examples to test domain independent/dependent text normalization.

3-4	*three to four*
	three four
<math_exp> 3-4 </math_exp>	*three minus four*

Some systems may not have a discrete level of orthographic or SNOR representation that easily lends itself to the type of evaluation described in this section. Such systems may have to evaluate their text normalization component in terms of LTS conversion.

An automated test framework for the LTS conversion analysis minimally includes a set of test words and their phonetic transcriptions for automated lookup and comparison tests. The problem is the infinite nature of language: there are always *new words* that the system does not convert correctly, and many of these will initially lack a transcription of record even to allow systematic checking. Therefore, a comprehensive test program for test of phonetic conversion accuracy needs to be paired with a data development effort. The data effort has two goals: to secure a continuous source of potential new words, such as a 24-hour newswire feed, and to maintain and construct an offline test dictionary, where reference transcriptions for new words are constantly created and maintained by human experts. This requirement illustrates the codependence of automated and manual aspects of evaluation. Different types and sources of vocabulary need to be considered separately, and they may have differing testing requirements, depending, again, on the nature of the particular system to be evaluated. For example, some systems have elaborate subsystems targeted specifically for name conversion. Such systems may depend on other kinds of preprocessing technologies, such as name identification modules, that might be tested independently.

The correct phonetic representation of a word usually depends on its sentence and even discourse contexts, as described in Section 14.6. Therefore, the adequacy of LTS conversion should not, in principle, be evaluated on the basis of isolated word pronunciations. However, a list of isolated word pronunciations is often used in LTS conversion because of its simplicity. Discourse contexts are, in general, difficult to represent unless specific applications and markup tags are available to the evaluation database. A reasonable compromise is to use a list of independent sentences with their corresponding phonetic representation for the evaluation of grapheme-to-phoneme conversion.

Error analysis should be treated as equally important as the evaluation itself. For example, if a confusability matrix shows that a given system frequently confuses central and schwa-like unstressed vowels, this may be viewed as less serious than other kinds of errors. Other subareas of LTS conversion that could be singled out for special diagnosis and testing include morphological analysis and stress placement. Of course, testing with phonemic transcriptions is the ultimate *unit test* in the sense that it contains nothing to insure that the correctly transcribed words, when spoken by the system's artificial voice and prosody, are intelligible or pleasant to hear. Phone transcription accuracy is, thus, a necessary but not a sufficient condition of quality.

14.10. CASE STUDY: FESTIVAL

The University of Edinburgh's *Festival* [3] has been designed to take advantage of modular subcomponents for various standard functions. Festival provides a complete text and phonetic analysis with modules organized in sequence roughly equivalent to Figure 14.1. Festival outputs speech of quality comparable to many commercial synthesizers. While default

routines are provided for each stage of processing, the system is architecturally designed to accept alternative routines in modular fashion, as long as the data transfer protocols are followed. This variant of the traditional TTS architecture is particularly attractive for commercial purposes (development, maintenance, testing, scalability) as well as research. Festival can be called in various ways with a variety of switches and filters, set from a variety of sanctioned programming and scripting languages. These control options are beyond the scope of this overview.

14.10.1. Lexicon

Festival employs phonemes as the basic sounding units, which are used not only as the atoms of word transcriptions in the lexicons, but also as the organizing principle for unit selection (see Chapter 16) in the synthesizer itself. Festival can support a number of distinct phone sets and it supports mapping from one to another. A phone defined in a set can have various associated phonological features, such as vowel, high, low, etc.

The Festival lexicon, which may contain several components, provides pronunciations for words. The *addenda* is an optional list of words that are unique to a particular user, document, or application. The addenda is searched linearly. The main system lexicon is expected to be large enough to require compression and is assumed to reside on a disk or other external storage. It is accessed via binary search. The lexical entry also contains POS information, which can be modified according to the preference of the system configurer. A typical lexical entry consists of the word key, a POS tag, and phonetic pronunciation (with stress and possible syllabification indicated in parentheses):

("walkers" N (((w ao) 1) ((k er z) 0)))

If the syllables structure is not shown with parentheses, a syllabification rule component can be invoked. Separate entry lines are used for words with multiple pronunciations and/or POS, which can be resolved by later processing.

14.10.2. Text Analysis

Festival has been partially integrated with research on the use of automatic identification of document and discourse structures. The discourse tagging is done by a separate component, called SOLE [11]. The tags produced by SOLE indicate features that may have relevance for pitch contour and phrasing in later stages of synthesis (see Chapter 15). These must be recognized and partially interpreted at the text analysis phrase. The SOLE tags tell Festival when the text is comparing or contrasting two objects, when it's referring to old or new information, when it's using a parenthetical or starting a new paragraph, etc., and Festival will decide, based on this information, that it needs to pause, to emphasize or deemphasize, to modify its pitch range, etc.

Additionally, as discussed in Section 14.3, when document creators have knowledge about the structure or content of documents, they can express the knowledge through an XML-based synthesis markup language. A document to be spoken is first analyzed for all such tags, which can indicate alternative pronunciations, semantic or quasi-semantic attributes (different uses of numbers by context for example), as well as document structures, such as explicit sentence or paragraph divisions. The kinds of information potentially supplied by the SABLE tags[7] are exemplified in Figure 14.7.

```
<SABLE>
    <SPEAKER NAME="male1">
    The boy saw the girl in the park <BREAK/> with the telescope.
    The boy saw the girl <BREAK/> in the park with the telescope.

    Good morning <BREAK /> My name is Stuart, which is spelled
    <RATE SPEED="-40%">
    <SAYAS MODE="literal">stuart</SAYAS> </RATE>
    though some people pronounce it
    <PRON SUB="stoo art">stuart</PRON>. My telephone number
    is <SAYAS MODE="literal">2787</SAYAS>.

    I used to work in <PRON SUB="Buckloo">Buccleuch</PRON> Place,
    but no one can pronounce that.
    </SPEAKER>
</SABLE>
```

Figure 14.7 A document fragment augmented with SABLE tags can be processed by the Festival system [3].

For untagged input, or for input inadequately tagged for text division (<BREAK/>), sentence breaking is performed by heuristics, similar to Algorithm 14.1, which observe whitespace, punctuation, and capitalization. A linguistic unit roughly equivalent to a sentence is created by the system for the subsequent stages of processing.

Tokenization is performed by system or user-supplied routines. The basic function is to recognize potentially speakable items and to strip irrelevant whitespace or other non-speakable text features. Note that some punctuation is retained as a feature on its nearest word.

Text normalization is implemented by token-to-word rules, which return a standard orthographic form that can, in turn, be input to the phonetic analysis module. The token-to-word rules have to deal with text normalization issues similar to those presented in Section 14.4. As part of this process, token-type-specific rule sets may be applied to disambiguate tokens whose pronunciations are highly context dependent. For example, a disambiguation routine may be required to examine context for deciding whether *St.* should be realized as *Saint* or *street*. For general English-language phenomena, such as numbers and various

[7] SABLE and other TTS markup systems are discussed further in Chapter 15.

symbols, a standard token-to-word routine is provided. One interesting feature of the Festival system is a utility for helping to automatically construct decision trees to serve text normalization rules, when system integrators can gather some labeled training data.

The linguistic analysis module for the Festival system is mainly a POS analyzer. An *n*-gram based trainable POS tagger is used to predict the likelihoods of POS tags from a limited set given an input sentence. The system uses both a priori probabilities of tags given a word and *n*-grams for sequences of tags. The basic underlying technology is similar to the work in [6] and is described in Section 14.5. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography, as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

14.10.3. Phonetic Analysis

The homograph disambiguation is mainly resolved by POS tags. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

If a word fails lexical lookup, LTS rules may be invoked. These rules may be created by hand, formatted as shown below:

```
( # [ c h ] C = /k/)      // ch at word start, followed by a consonant, is /k/, e.g.,
Chris
```

Alternatively, LTS rules may be constructed by automatic statistical methods, much as described in Section 14.8 above, where CART LTS systems were introduced. Utility routines are provided to assist in using a system lexicon as a training database for CART rule construction.

In addition, Festival system employs *post-lexical rules* to handle *context coarticulation*. Context coarticulation occurs when surrounding words and sounds, as well as speech style, affect the final form of pronunciation of a particular phoneme. Examples include reduction of consonants and vowels, phrase final devoicing, and *r*-insertion. Some coarticulation rules are provided for these processes, and users may also write additional rules.

14.11. Historical Perspective and Further Reading

Text-to-speech has a long and rich history. You can hear samples and review almost a century's worth of work at the Smithsonian's Speech Synthesis History Project [19]. A good source for multilingual samples of various TTS engines is [20].

The most influential single published work on TTS has been *From Text to Speech: The MITalk System* [1]. This book describes the MITalk system, from which a large number

of research and commercial systems were derived during the 1980s, including the widely used DECTalk system [9]. The best compact overall historical survey is Klatt's *Review of Text-to-Speech Conversion for English* [15]. For deeper coverage of more recent architectures, refer to [7]. For an overview of some of the most promising current approaches and pressing issues in all areas of TTS and synthesis, see [30]. One of the biggest upcoming issues in TTS text processing is the architectural relation of specialized TTS text processing as opposed to general-purpose natural language or document structure analysis. One of the most elaborate and interesting TTS-specific architectures is the multilingual text processing engine described in [27]. This represents a commitment to providing exactly the necessary and sufficient processing that speech synthesis requires, when a general-purpose language processor is unavailable.

However, it is expected that natural language and document analysis technology will become more widespread and important for a variety of other applications. To get an idea of what capabilities the natural language analysis engines of the future may incorporate, refer to [12] or [2]. Such generalized engines would serve a variety of clients, including TTS, speech recognition, information retrieval, machine translation, and other services which may seem exotic and isolated now but will increasingly share core functionality. This convergence of NL services can be seen in a primitive form today in Japanese *input method editors* (IME), which offload many NL analysis tasks from individual applications, such as word processors and spreadsheets, and unify these functions in a single common processor [18].

For letter-to-sound rules, NETalk [25], which describes automatic learning of LTS processes via neural network, was highly influential. Now, however, most systems have converged on decision-tree systems similar to those described in [14].

REFERENCES

[1] Allen, J., M.S. Hunnicutt, and D.H. Klatt, *From Text to Speech: the MITalk System*, 1987, Cambridge, UK, University Press.

[2] Alshawi, H., *The Core Language Engine*, 1992, Cambridge, US, MIT Press.

[3] Black, A.W., P. Taylor, and R. Caley, "The Architecture of the Festival Speech Synthesis System," *3rd ESCA Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, University of Edinburgh, pp. 147-151.

[4] Boguraev, B. and E.J. Briscoe, *Computational Lexicography for Natural Language Processing*, 1989, London, Longmans.

[5] Chomsky, N. and M. Halle, *The Sound Patterns of English*, 1968, Cambridge, MIT Press.

[6] Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proc. of the Second Conf. on Applied Natural Language Processing*, 1988, Austin, Texas, pp. 136-143.

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

[8] Francis, W. and H. Kucera, *Frequency Analysis of English Usage*, 1982, New York, N.Y., Houghton Mifflin.

[9] Hallahan, W.I., "DECtalk Software: Text-to-Speech Technology and Implementation," *Digit Technical Journal*, 1995, 7(4), pp. 5-19.

[10] Higgins, J., *Homographs*, 2000, http://www.stir.ac.uk/celt/staff/higdox/wordlist/homogrph.htm.

[11] Hitzeman, J., *et al.*, "On the Use of Automatically Generated Discourse-Level Information in a Concept-to-Speech Synthesis System," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 2763-2766.

[12] Jensen, K., G. Heidorn, and S. Richardson, *Natural Language Processing: the PLNLP Approach*, 1993, Boston, MASS, Kluwer Academic Publishers.

[13] Jiang, L., H.W. Hon, and X. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Proc. of Eurospeech*, 1997, Rhodes, Greece, pp. 605-608.

[14] Jiang, L., H.W. Hon, and X.D. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Eurospeech'97*, 1997, Rhodes, Greece.

[15] Klatt, D., "Review of Text-to-Speech Conversion for English," *Journal of Acoustical Society of America*, 1987, **82**, pp. 737-793.

[16] LDC, *Linguistic Data Consortium*, 2000, http://www.ldc.upenn.edu/ldc/noframe.html.

[17] Levine, J., Mason, T., Brown, D., *Lex and Yacc*, 1992, Sebastopol, CA, O'Rielly & Associates.

[18] Lunde, K., *CJKV Information Processing Chinese, Japanese, Korean & Vietnamese Computing*, 1998, O'Reilly.

[19] Maxey, H., *Smithsonian Speech Synthesis History Project*, 2000, http://www.mindspring.com/~dmaxey/ssshp/.

[20] Möhler, G., *Examples of Synthesized Speech*, 1999, http://www.ims.uni-stuttgart.de/phonetik/gregor/synthspeech/.

[21] Nye, P.W., *et al.*, "A Plan for the Field Evaluation of an Automated Reading System for the Blind," *IEEE Trans. on Audio and Electroacoustics*, 1973, **21**, pp. 265-268.

[22] OMF, *CML - Chemical Markup Language*, 1999, http://www.xml-cml.org/.

[23] Richardson, S.D., W.B. Dolan, and L. Vanderwende, "MindNet: Acquiring and Structuring Semantic Information from Text," *ACL'98: 36th Annual Meeting of the Assoc. for Computational Linguistics and 17th Int. Conf. on Computational Linguistics*, 1998, pp. 1098-1102.

[24] Sable, *The Draft Specification for Sable version 0.2*, 1998, http://www.cstr.ed.ac.uk/projects/sable_spec2.html.

[25] Sejnowski, T.J. and C.R. Rosenberg, *NETtalk: A Parallel Network that Learns to Read Aloud*, 1986, Johns Hopkins University.

[26] Sluijter, A.M.C. and J.M.B. Terken, "Beyond Sentence Prosody: Paragraph Intonation in Dutch," *Phonetica*, 1993, **50**, pp. 180-188.

[27] Sproat, R., *Multilingual Text-To-Speech Synthesis: The Bell Labs Approach*, 1998, Dordrecht, Kluwer Academic Publishers.

[28] Sproat, R. and J. Olive, "An Approach to Text-to-Speech Synthesis," in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Amsterdam, pp. 611-634, Elsevier Science.

[29] Turing, A.M., "Computing Machinery and Intelligence," *Mind*, 1950, LIX(236), pp. 433-460.

[30] van Santen, J., *et al.*, *Progress in Speech Synthesis*, 1997, New York, Springer-Verlag.

[31] van Santen, J., *et al.*, "Report on the Third ESCA TTS Workshop Evaluation Procedure," *Third ESCA Workshop on Speech Synthesis*, 1998, Sydney, Australia.

[32] Vitale, T., "An Algorithm for High Accuracy Name Pronunciation by Parametric Speech Synthesizer," *Computational Linguistics*, 1991, 17(3), pp. 257-276.

[33] W3C, *Aural Cascading Style Sheets (ACSS)*, 1997, http://www.w3.org/TR/WD-acss-970328.

[34] W3C, *W3C's Math Home Page*, 1998, http://www.w3.org/Math/.

[35] W3C, *Extensible Markup Language (XML)*, 1999, http://www.w3.org/XML/.

[36] Wall, L., Christiansen, T., Schwartz, R., *Programming Perl*, 1996, Sebastopol, CA, O'Rielly & Associates.

INDEX

B

D

E

F

G

N

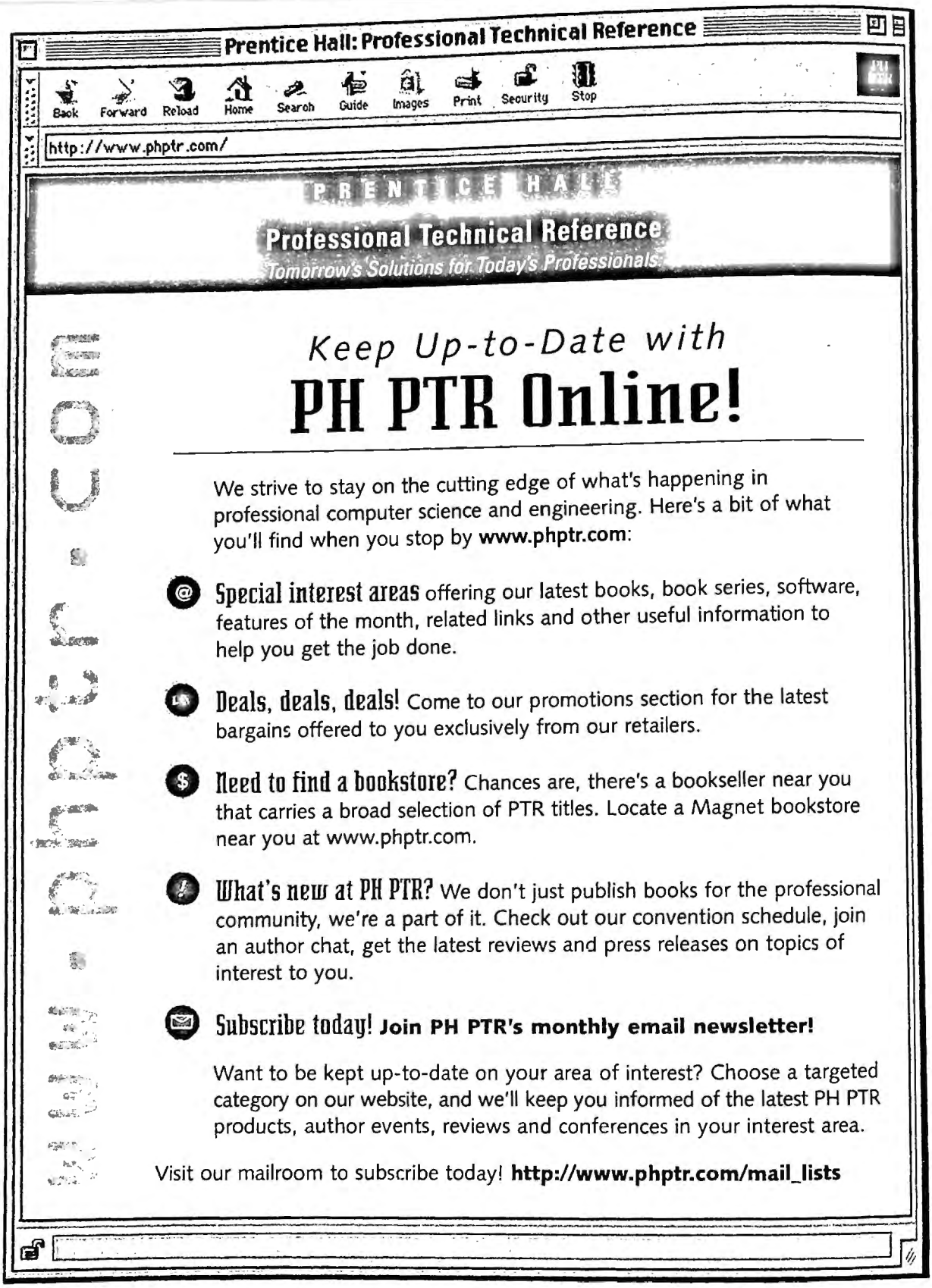
O

Q

R

S

T


Prentice Hall: Professional Technical Reference
Back
Forward
Reload
Home
Search
Guide
Images
Print
Security
Stop
http://www.phptr.com/
PRENTICE HALL
Professional Technical Reference
Tomorrow's Solutions for Today's Professionals
Keep Up-to-Date with
PH PTR Online!
We strive to stay on the cutting edge of what's happening in professional computer science and engineering. Here's a bit of what you'll find when you stop by www.phptr.com:
Special interest areas offering our latest books, book series, software, features of the month, related links and other useful information to help you get the job done.
Deals, deals, deals! Come to our promotions section for the latest bargains offered to you exclusively from our retailers.
Need to find a bookstore? Chances are, there's a bookseller near you that carries a broad selection of PTR titles. Locate a Magnet bookstore near you at www.phptr.com.
What's new at PH PTR? We don't just publish books for the professional community, we're a part of it. Check out our convention schedule, join an author chat, get the latest reviews and press releases on topics of interest to you.
Subscribe today! Join PH PTR's monthly email newsletter!
Want to be kept up-to-date on your area of interest? Choose a targeted category on our website, and we'll keep you informed of the latest PH PTR products, author events, reviews and conferences in your interest area.
Visit our mailroom to subscribe today! http://www.phptr.com/mail_lists

Voice/Speech Recognition Technology

XUEDONG HUANG

ALEX ACERO

HSIAO-WUEN HON

SPOKEN LANGUAGE PROCESSING

A Guide to Theory, Algorithm, and System Development

- New advances in spoken language processing: *theory and practice*
- In-depth coverage of speech processing, speech recognition, speech synthesis, spoken language understanding, and speech interface design
- Many case studies from state-of-the-art systems, including examples from Microsoft®'s advanced research labs

Spoken Language Processing draws on the latest advances and techniques from multiple fields: computer science, electrical engineering, acoustics, linguistics, mathematics, psychology, and beyond. Starting with the fundamentals, it presents all this and more:

- Essential background on speech production and perception, probability and information theory, and pattern recognition
- Extracting information from the speech signal: useful representations and practical compression solutions
- Modern speech recognition techniques: hidden Markov models, acoustic and language modeling, improving resistance to environmental noises, search algorithms, and large vocabulary speech recognition
- Text-to-speech: analyzing documents, pitch and duration controls, trainable synthesis, and more
- Spoken language understanding: dialog management, spoken language applications, and multimodal interfaces

To illustrate the book's methods, the authors present detailed case studies based on state-of-the-art systems, including Microsoft's Whisper speech recognizer, Whistler text-to-speech system, Dr. Who dialog system, and the MiPad handheld device. Whether you're planning, designing, building, or purchasing spoken language technology, this is the state of the art—from algorithms through business productivity.

About the Authors

XUEDONG HUANG is founder and head of the Speech Technology Group at Microsoft Research. He received his Ph.D. from the University of Edinburgh. He is an IEEE Fellow.

ALEX ACERO and HSIAO-WUEN HON are Senior Researchers at Microsoft Research and Senior Members of IEEE. Both received doctorates from Carnegie Mellon University.

PRENTICE HALL
Upper Saddle River, NJ 07458
www.phptr.com

ISBN 0-13-022616-5
90000
9 780130 226167