



MICHIGAN STATE UNIVERSITY LIBRARIES



3 1293 00885 0194

This is to certify that the

dissertation entitled

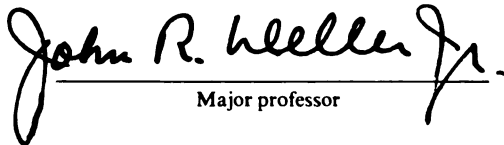
**Large-Vocabulary Continuous-Speech Recognition  
Using Partitioned Graph Search**

presented by

Chuang-Chien Chiu

has been accepted towards fulfillment  
of the requirements for

Ph.D. degree in Electrical  
Engineering

  
Major professor

Date August 2, 1993.



PLACE IN RETURN BOX to remove this checkout from your record.  
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

MSU Is An Affirmative Action/Equal Opportunity Institution

c:\circ\dstdue.pm3-p.1

**LARGE-VOCABULARY CONTINUOUS-  
SPEECH RECOGNITION USING  
PARTITIONED GRAPH SEARCH**

By

*Chuang-Chien Chiu*

**A DISSERTATION**

Submitted to  
Michigan State University  
in partial fulfillment of the requirements  
for the degree of

**DOCTOR OF PHILOSOPHY**

Department of Electrical Engineering

1993

**ABSTRACT**

**LARGE-VOCABULARY CONTINUOUS-  
SPEECH RECOGNITION USING  
PARTITIONED GRAPH SEARCH**

By

*Chuang-Chien Chiu*

Contemporary speech recognition ordinarily involves the decoding of an utterance by association of a set of acoustic observations with a best path in a finite-state network. The speech recognition problem is then reduced to a network search problem. Conventional left-to-right search methods generally encounter two major problems which make them infeasible and unattractive: First is the computational complexity. Once the network becomes large, a complete left-to-right search of the network is computationally impractical, even when a pruning strategy is applied. The second problem is in the presence of noise. If the search uses the data in a sequential left-to-right manner, the correct path through the network is likely to be “derailed” or curtailed by nonstationary sources of noise typical of many practical environments where speech recognition is desirable.

This research is concerned with a novel method called *partitioned graph search* which avoids the drawbacks of conventional search techniques. The *partitioned graph search* techniques employ a computationally inexpensive pass to pare down the problem to one involving a smaller graph of likely solutions. The smaller graph can then be subjected to intense scrutiny to find an optimal solution.

Three major contributions have been made in this work: (1) A vertex partitioning algorithm for generally nonplanar graphs is developed, and the key operation for finding a cycle for planar graph subpartitioning is improved; (2) A two-pass search

algorithm, which is applicable to the case of unknown boundaries in the observation string, is developed; (3) The partitioned graph search techniques are applied to recognition of continuous-speech taken from the TIMIT speech database. A language graph with perplexity 5.1 containing 14,259 vertices and 28,838 edges is constructed from this database. Comparisons of the results using three different search approaches are made. These experiments include conventional left-to-right search, random selection of as many vertices as in the partitioned graph search cases for evaluation, and partitioned graph search.

**Copyright by**  
**CHUANG-CHIEN CHIU**  
**1993**

*To my parents*

*Chia-Jen Chiu and Shih-Mey Liang*



# Acknowledgments

First and foremost, I would like to thank my thesis advisor, Professor John R. Deller, Jr., for his continuous encouragement, patient guidance and generous support throughout my graduate study at Michigan State University. Without his help, this work could not possibly be finished by now. I would also like to thank my “teacher mother” - Joan, for her tolerance of my occupying her husband’s time, and for her kindness to me in these years. They have made my overall learning experience enjoyable in the Speech Processing Lab at Michigan State University.

I would like to thank all the members in my Ph.D. guidance committee, Professor Abdol-Hossein Esfahanian (Department of Computer Science), Professor Majid Nayeri (Department of Electrical Engineering) and Professor Clifford E. Weil (Department of Mathematics) for their time and effort in discussing and reviewing my thesis. I am also deeply indebted to Dr. C.G. Venkatesh in CTA Incorporated, who initiated the idea for using graph partitioning techniques in signal decoding, and helped me to shape many aspects in graph partitioning algorithms of this work.

I would like to express my gratitude to everyone in my family for their love and support. I am especially grateful to my parents for all the sacrifices they made in my upbringing. I also owe a great deal to the friends I met here at Michigan State University. Most of all, I wish to thank my host family, Wellington, Emily and Millicent Ow for their friendliness and kindness during the past four years.

A very special word of thanks for my fiancée, Eun-Hi Park, for all her love, patience, encouragement and understanding over past three and a half years we have

known each other. Because of her, I feel more hopeful in the future. Also, I appreciate her promise to be my wife in this coming September. Therefore, I can have a good chance in my lifetime to repay her.

This work is supported in part by a contract from CTA, Incorporated and by an Ameritech Faculty Fellowship awarded to Dr. Deller.

# Contents

<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction and Background</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.1.1 The Significance of Partitioned Graph Search . . . . .	1
1.1.2 Search Techniques in Speech Recognition . . . . .	2
1.1.3 General Scope and Achievements . . . . .	4
1.2 Partitioned Graph Search . . . . .	9
1.2.1 Graph Partitioning . . . . .	9
1.2.1.1 Planar Separator Theorems . . . . .	9
1.2.1.2 A Revised PST for General Graphs . . . . .	12
1.2.2 Graph Search in the Presence of Partitioning . . . . .	13
1.2.2.1 First Pass: Modified Stack Decoding . . . . .	13
1.2.2.2 Second Pass: Optimal Solution . . . . .	14
<b>2 Algorithms for Graph Partitioning</b>	<b>16</b>
2.1 Introduction . . . . .	16
2.2 Building a Language Graph . . . . .	17
2.2.1 The Need for a Large Graph . . . . .	17
2.2.2 The Method for Building the Language Graph . . . . .	20
2.3 Graph Partitioning Algorithms . . . . .	23
2.3.1 Partitioning Algorithms for Planar Graphs . . . . .	23
2.3.2 Partitioning Nonplanar Graphs . . . . .	25
2.4 Cycle-Finding Algorithm . . . . .	28
2.4.1 Edge-Scanning Approach . . . . .	28
2.4.2 An Example . . . . .	33
<b>3 A Stack Algorithm for Partitioned Search</b>	<b>42</b>
3.1 The $n$ -Best Strategy . . . . .	42
3.2 First Pass: Modified Stack Decoding . . . . .	44
3.2.1 Vertex Evaluation Algorithm . . . . .	44
3.2.1.1 Baseline Algorithm . . . . .	44
3.2.1.2 Refining Boundaries Following Vertex Evaluation . . . . .	47

3.2.2	Modified Stack Decoding . . . . .	52
3.2.3	Computation Savings from the First Pass . . . . .	56
3.3	Second Pass: Optimal Solution . . . . .	58
<b>4</b>	<b>Implementation and Experimental Issues</b>	<b>59</b>
4.1	The TIMIT Database . . . . .	59
4.2	Implementation Issues . . . . .	63
4.2.1	Building a Language Graph . . . . .	63
4.2.2	Speech Modeling for Partitioned Graph Search Techniques . .	68
4.2.3.1	Cepstral Analysis . . . . .	68
4.2.3.2	Vector Quantization . . . . .	69
4.2.3.3	HMM Training . . . . .	72
4.2.3	Graph Partitioning . . . . .	74
4.2.4	Two-Pass Graph Search . . . . .	79
4.3	Experiments . . . . .	80
4.3.1	Methods and Measures of Performance . . . . .	80
4.3.2	Results and Discussion . . . . .	82
<b>5</b>	<b>Further Discussion, Conclusions and Future Work</b>	<b>91</b>
5.1	Summary and Further Discussion . . . . .	91
5.2	Contributions . . . . .	93
5.3	Future Work . . . . .	94
	<b>Appendix A: Graph-Theoretic Notations and Definition</b>	<b>97</b>
	<b>Appendix B: Supporting Lemmas for the PST</b>	<b>100</b>
	<b>Appendix C: Lipton-Tarjan Approach to Finding a Cycle</b>	<b>103</b>
	<b>Appendix D: A Planarity Testing Algorithm</b>	<b>106</b>
	<b>Appendix E: Elements of the Hidden Markov Model</b>	<b>109</b>
	<b>Bibliography</b>	<b>115</b>

# List of Tables

2.1	The set of vertices in each face of the planar embedding of the graph in Fig. 2.13. . . . .	36
2.2	The vertices on each level of the graph in Fig. 2.13 after running BFS. . . . .	36
2.3	The set of vertices in each face of the planar embedding of the newly formed graph. . . . .	36
2.4	The set of vertices in each face of $H'$ except the vertices which are on the chosen cycle. . . . .	37
2.5	The set of vertices on each side of the chosen cycle. . . . .	40
4.6	Summary of the usage of the TIMIT database in this work. . . . .	60
4.7	Phonetic labels used in the TIMIT database. . . . .	61
4.8	No. of phonetic segments in the training set. . . . .	73
4.9	A list of function words appearing in this research. . . . .	75
4.10	Different ways the function word <i>the</i> is pronounced in TIMIT. . . . .	75
4.11	Different ways the function word <i>in</i> is pronounced in TIMIT. . . . .	76
4.12	Different ways the function word <i>with</i> is pronounced in TIMIT. . . . .	76
4.13	Different ways the function word <i>a</i> is pronounced in TIMIT. . . . .	76
4.14	Experimental results. . . . .	83
4.15	Experimental results with three problematic test utterances removed. . . . .	86

# List of Figures

1.1	An utterance with the boundaries for each word marked. . . . .	8
1.2	The block diagram of the speech recognition system using the partitioned graph search techniques. . . . .	9
1.3	The condition implied by the Planar Separator Theorem. . . . .	10
2.4	Two possible graphs which can “generate” sentences 1 – 3 listed in the text. . . . .	18
2.5	An example of a left-to-right 6-state model. . . . .	20
2.6	An example of a partial language graph. . . . .	22
2.7	The condition implied by the PST of Lipton and Tarjan. . . . .	25
2.8	The condition implied by the PST of Djidjev. . . . .	26
2.9	Three cases cause the violation of the planar separator theorem. . . .	28
2.10	The block diagram for finding the partitioning cycle in the new graph.	30
2.11	The block diagram for the ONE module. . . . .	31
2.12	The block diagram for the ANE module. . . . .	34
2.13	An example graph used to illustrate the subpartitioning algorithm. .	34
2.14	The planar embedding of the planar graph of Fig. 2.13. . . . .	35
2.15	The planar embedding of the newly formed graph. . . . .	37
2.16	A desired cycle is found in this example. . . . .	38
2.17	The visual understanding of the partitioning results. . . . .	41
3.18	The $n$ -best search paradigm. . . . .	42
3.19	An example illustrates how the phone models can be combined to form a big word model. . . . .	47
3.20	Three possible cases of merging potential intervals after word spotting operation. . . . .	49
3.21	A conceptual plot of the word-spotting results for the selected vertices.	51
4.22	The datafile structure of the TIMIT database. . . . .	62
4.23	All the possible vertices with different phonetic labels for the word <i>ask</i> in a large graph. . . . .	65
4.24	The word <i>ask</i> in a bigram graph. . . . .	65
4.25	An example of a language graph. . . . .	66
4.26	Filter bank for generating mel-based cepstral coefficients. . . . .	70
4.27	The complexity measure (CM) for normal speech using three different methods. . . . .	87
4.28	The complexity measure (CM) for normal speech using partition and random selection methods. . . . .	88

4.29	The complexity measure (CM) for noisy speech using three different methods. . . . .	88
4.30	The complexity measure (CM) for noisy speech using partition and random selection methods. . . . .	89
4.31	The overall performance as measured by the normalized word accuracy (NWA) of three different search methods. . . . .	90
A.32	A plane graph with four faces. . . . .	98
C.33	Cases of Step 4 in the L&T approach for finding a proper cycle. . . .	103
D.34	The planarity testing algorithm of Demoucron <i>et al.</i> . . . .	108
E.35	Some examples of four-state HMM topologies. . . . .	112

# Chapter 1

## Introduction and Background

---

### 1.1 Introduction

#### 1.1.1 The Significance of Partitioned Graph Search

Contemporary speech recognition ordinarily involves the decoding<sup>1</sup> of an utterance by association of a set of acoustic observations with a best path in a finite-state network. Therefore, the speech recognition problem is reduced to a network search problem. However, almost all existing search algorithms, which are related to either *breadth-first search* or *depth-first search (best-first search)* [1], [2], [3], [4], implement a left-to-right search strategy. These conventional search methods encounter two major problems which make them infeasible and unattractive: First is the computational complexity. Once the network becomes large (typically  $10^6 - 10^{11}$  [5]), a complete left-to-right search of the network is computationally impractical, even when a pruning strategy is applied. The second problem is the presence of noise. If the search uses the data in a sequential left-to-right manner, the correct path through the network

---

<sup>1</sup>A decoding problem involves the finding of the most likely path in a graph, say  $\mathbf{G}$ , given an observation sequence, say  $\mathbf{Y}$ , and an *a priori* structure embedded in  $\mathbf{G}$ .



is likely to be “derailed” or curtailed by impulsive noise typical of many practical environments where speech recognition is desirable. For example, if noisy speech is to be recognized, and if the “left-most” observations are corrupted by noise, the left-to-right search might immediately eliminate the correct path.

The impetus for this research is the need for robust search methods which avoid the drawbacks of conventional search techniques. A novel method called *partitioned graph search* is presented. Partitioned graph search offers a systematic way of locating a very small number of vertices which are guaranteed to give effective coverage of the graph thus assuring high pruning safety. Briefly, a computationally inexpensive pass is employed to pare down the problem to one involving a smaller graph of likely solutions from which the ultimate solution is quickly found. The techniques are used to undertake a large, complex recognition task based on graph partitioning and network searching techniques which achieve the reduction of computational load and improve the recognition rate with respect to that of left-to-right approaches. The purpose of this dissertation is to describe significant progress towards this goal.

### **1.1.2 Search Techniques in Speech Recognition**

Among existing search algorithms for speech recognition tasks, variations of the *Viterbi search* algorithm [6] and the *stack decoding* algorithm [7] are the most popular. Viterbi search examines many possible paths in parallel by progressing through the network. It is based on the *principle of optimality* [8], and has been used extensively in dynamic-programming-based speech recognition. There are a number of signifi-

cant bookkeeping issues involved in managing the complexity of the Viterbi search algorithm. Briefly, the Viterbi search is a time-synchronous search algorithm that completely processes time  $t$  before going on to time  $t + 1$ . Each state for time  $t$  is updated by the best score from states at time  $t - 1$ . In this fashion, the most likely state sequence can be recovered at the end of the search.

A common goal of fast search algorithms is the reduction of the search space which must be evaluated. Some measure of goodness of a path is established. Paths that fail the goodness test are pruned (not extended). The pruning approach is used to avoid the combinatorial explosion of paths which accompanies a full search. A full search is efficient enough for a smaller task using the Viterbi strategy. However, for a large task, Viterbi can be very time and space consuming. Therefore, the *beam search* [8], [9] technique is used to prune the search space. The basic principle of Viterbi beam search is that at any time, all paths whose probabilities fall within some threshold of the best global path are retained. Note that Viterbi beam search only guarantees a locally optimum path with the risk that the correct global path may be overlooked [8], [10]. In spite of the reduction in the search space achieved by Viterbi beam search, most continuous-speech recognition tasks are still intractable. In order to reduce the amount of computation required by the Viterbi search algorithm in a large network, *stack decoding* is frequently adopted.

Stack decoding, first proposed by Jelinek in 1969 [7], is also a left-to-right search approach which starts at the initial state and extends paths of different lengths. Thus, it is not a “time-synchronous” search technique [8]. Stack decoding is an implementation of the best-first tree search [11]. The term “stack” refers to an ordered

set held in the decoder's memory. The following must be contained in each stack entry:

1. The history of a partial path.
2. The partial path likelihood (which is used to order the entries).
3. An end-of-path flag (if appropriate).

To carry out best-first search in speech recognition tasks, a likelihood function must be defined with which to compare incomplete paths of varying lengths. It is also desired to have a likelihood function which increases along the most likely path, and decreases along other paths. Stack decoding will be incorporated with partitioned graph search techniques in our continuous-speech recognition task. A modified version of stack decoding will be discussed in detail in Chapter 3.

### 1.1.3 General Scope and Achievements

In previous work, Venkatesh *et al.* [13] presented the conceptual development of a graph-theoretic strategy for reducing the computational complexity of signal decoding with respect to conventional decoding approaches [6], [11], [14], [15]. The technique, which is based on the “Planar Separator Theorem” of Lipton and Tarjan [17], uses a partitioning approach to locate  $\mathcal{O}(\sqrt{8N})$  vertices for evaluation from an  $N$ -vertex decoding graph<sup>2</sup>  $\mathbf{G}$ . In the partitioned graph search method, evaluation of this relatively small number of vertices is used to find an optimal path for a given observation sequence. Although many of the basic conceptual ideas underlying this method were

---

<sup>2</sup>In speech recognition task, the decoding graph  $\mathbf{G}$  is a finite-state network which represents all “linguistic” knowledge about a corpus of speech data.

worked out by Venkatesh *et al.*, no specific computer algorithms were developed for selecting these “high payoff” vertices of  $G$ , and no applications were demonstrated.

In [16], Chiu reports two significant extensions of the Venkatesh research. One is the development of a planar graph partitioning algorithm to select<sup>3</sup>  $\mathcal{O}(\sqrt{N})$  vertices for evaluation from an  $N$ -vertex planar decoding graph. This algorithm is based on the solution of Djidjev [18] whose separator<sup>4</sup> is of size  $\mathcal{O}(\sqrt{6N})$ . In the process, a method for finding an appropriate simple cycle to complete the vertex partitioning is developed as well. Recently, the algorithm for finding the cycle has been improved [20], and the result will be discussed in more detail in Chapter 2. The other contribution in [16] is to test the partitioned graph search technique in an experiment in which the time boundaries in the observation sequence are known. The experiment involves a language graph (the method to construct such a language graph was presented in [16]) consisting of 1,000 vertices and 1,200 edges using an early version of a “multiple stack” decoding algorithm developed by Deller [13]. It has been shown that there is the potential for a steep decrease in the computational complexity using the partitioned graph search approach when compared with conventional left-to-right decoding.

The present research focuses on applying partitioned graph search techniques to recognize *continuous* speech. There are four principal problems to be investigated. Three of them are related to the theoretical and implementational issues of the par-

---

<sup>3</sup>Since  $N$  is usually much larger than 6 or 8, we henceforth write  $\mathcal{O}(\sqrt{N})$  to indicate the separator size.

<sup>4</sup>A *separator* is a set of vertices or edges whose removal disconnects the graph into two subgraphs each of which contains no more than some constant fraction of the original weight.

titioned graph search. The fourth is concerned with experimental issues arising when the partitioned search is applied to real speech data. The major problems and their significance are as follows:

1. Certain practical difficulties must be considered in implementing the partitioning method. The most obvious problem is that one cannot impose an *a priori* condition of planarity to a graph in order to get an  $\mathcal{O}(\sqrt{N})$  separator according to the planar separator theorem [17], [18]. The graph to be partitioned and searched will often be nonplanar in practice. Thus, the extension of the existing planar partitioning technique for generally nonplanar graphs is very significant.
2. In continuous-speech recognition tasks, the boundaries in the observation sequence are generally unknown. For example, Fig. 1.1 illustrates the boundaries for each word in the utterance. Without knowledge of the boundaries for each word in an utterance, the recognition task becomes significantly more challenging. Thus, the extension of previous search algorithms to the unknown boundaries case is very important.
3. In a large graph search problem, a subgraph remains after the first partition and search, which can then be further partitioned and searched in a similar manner if the remaining subgraph is still large. An important problem is how to perform the repartition and search to find the most likely path. If this procedure can be done, the solution would be expected to converge rapidly since each partition and search involves  $\mathcal{O}(\sqrt{N})$  or fewer evaluations. Because of the relatively small graph (14,259 vertices) employed in this work, this issue will only be explored

theoretically and superficially in the experiments.

The major problem to be addressed by experimentation is as follows:

4. When partitioned graph search is applied to real speech data, two very important issues must be considered: One is to test performance of partitioned search with respect to conventional left-to-right methods. The other is to test robustness of the search to intermittent noise. Added robustness to noise is shown to be a principal benefit of the method. Generally, the problem is: Is the partitioned graph search approach more robust than conventional left-to-right methods in noisy environments? This issue is significant since communication environments are frequently corrupted by noise. A comparison of sensitivity to noise and computational complexity among partitioned search and the conventional left-to-right approaches is reported in this work.

The major contributions of this research are summarized as follows:

1. A vertex partitioning algorithm for generally nonplanar graphs is developed, and the key operation for finding a cycle for the planar graph subpartitioning is improved.
2. A partitioned graph search algorithm, which is applicable to the case of unknown boundaries in the observation string, is developed.
3. Partitioned graph search techniques are applied to recognition of continuous-speech taken from the TIMIT [21] database. Comparisons of the results using the following approaches for recognizing continuous speech are made:

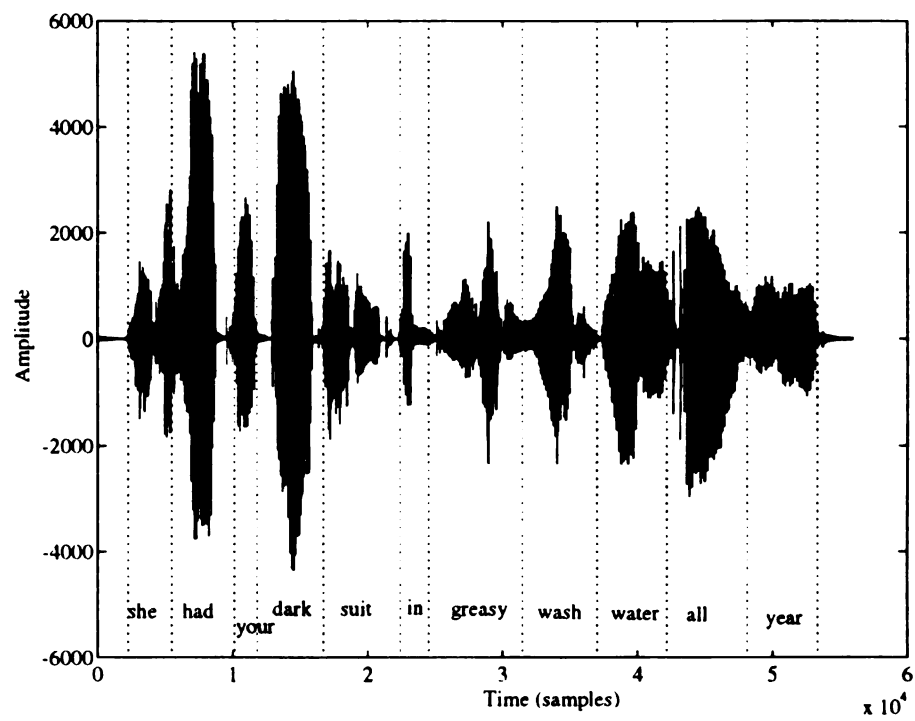


Figure 1.1: An utterance with the boundaries for each word marked. The sample rate on the data is 16 kHz.

- (a) partitioned graph search.
- (b) left-to-right search using *random* selection of as many vertices as in the partitioned graph search cases for evaluation.
- (c) conventional left-to-right stack search with pruning.

The block diagram of the speech recognition system using the partitioned graph search techniques is shown in Fig. 1.2. There are essentially four steps in the recognition process involving three externally generated datasets. The implementation of Fig. 1.2 is described in detail in Chapter 4.

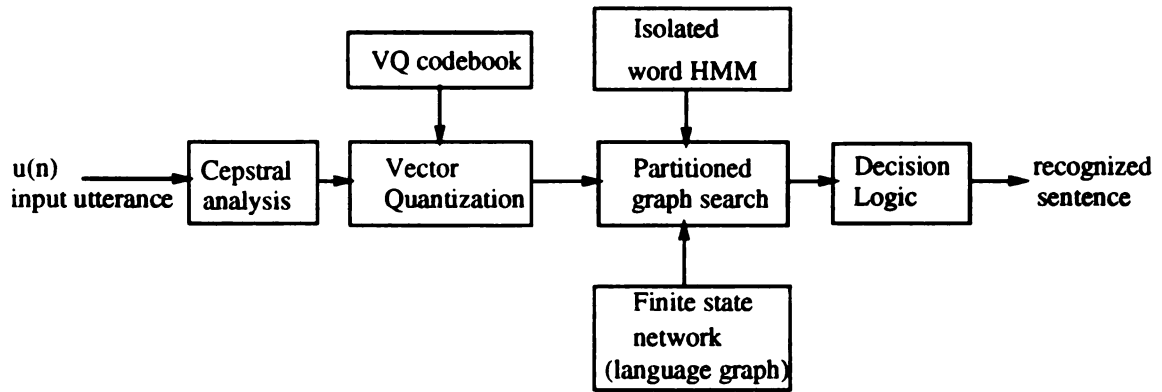


Figure 1.2: The block diagram of the speech recognition system using partitioned graph search.

## 1.2 Partitioned Graph Search

### 1.2.1 Graph Partitioning

#### 1.2.1.1 Planar Separator Theorem

Partitioned graph search employs a computationally inexpensive pass to pare down the problem to one involving a smaller graph of likely solutions. The smaller graph can



then be subjected to intense scrutiny to find an optimal solution. The salient features of the partitioning procedure are best understood by overviewing the underlying theory and discussing its advantages with respect to conventional left-to-right search. The following theorem, which is a general statement of the *Planar Separator Theorem* (PST), is fundamental to the methods:

**Theorem 1.1** *Let  $G$  be any planar graph with  $N$  vertices having non-negative vertex rewards summing to no more than one. Then  $G$  has a set of vertices  $C$  of size  $\mathcal{O}(\sqrt{N})$  which separates the set of vertices  $A$  from the set of vertices  $B$ , where  $A$ ,  $B$ , and  $C$  comprise a partition of the vertices in the given planar graph and neither  $A$  nor  $B$  has total reward exceeding  $2/3$ . The set  $C$  is called an  $\mathcal{O}(\sqrt{N})$  separator.*

Without loss of generality, one may assume that the planar graphs considered here are connected. The theorem is also true for planar graphs which are not connected [17]. The condition implied by the PST is illustrated in Fig. 1.3.

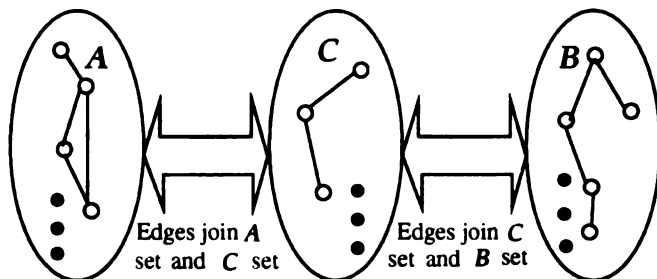


Figure 1.3: The condition implied by the Planar Separator Theorem.  $A$ ,  $B$ ,  $C$  form a vertex partition in a planar graph such that no edge joins a vertex in  $A$  with a vertex in  $B$ .

The vertices in the set  $C$  separate the graph into two sets of vertices  $A$  and  $B$  making it impossible to pass from one to the other without encountering the set  $C$ . Therefore, the vertices in the set  $C$  must contain many convergent and divergent

paths. Loosely speaking, the vertices of the set  $\mathcal{C}$  can be considered “bottlenecks” in the graph into which many paths converge. The PST therefore, guarantees the selection of significant vertices (those vertices in the set  $\mathcal{C}$ ) which will cover many paths.

One might select, in some *ad hoc* manner, vertices with large numbers of entering and exiting edges. It is known that these vertices represent “bottlenecks” in the graph upon which many paths converge. However, the location of  $\mathcal{O}(\sqrt{N})$  such vertices in no way assures proper coverage of the graph and the process cannot be done efficiently when  $N$  is large. By *coverage* we mean the extent to which the selected vertices represent a significant number of the paths, ideally with multiple vertices per each path; and, to a lesser extent, the degree to which the selected vertices are widely distributed throughout the graph so that better statistical use can be made of the observation sequence.

The increased coverage provided by the partitioning procedure will generally provide an acceptable “pruning safety.” On the other hand, the set  $\mathcal{C}$  is relatively small [ $\mathcal{O}(\sqrt{N})$ ], and these vertices are selected at distributed locations throughout  $\mathcal{G}$ , rather than always “from the left” which is conventional [11], [14]. Thus, the use of this set can minimize the number of vertex evaluations. If the procedure for evaluating vertices is very costly, the partitioning approach will greatly improve the computational cost compared to a conventional left-to-right search strategy.

For partitioning, there are two algorithms available. One is based on the PST of Lipton and Tarjan [17], the other is based on the theorem of Djidjev [18]. The issue of which of these is better for partitioning the graph in solving the continuous-speech

recognition problem is addressed in Chapter 2.

### 1.2.1.2 A Revised PST for General Graphs

Another major issue for the present project involves the application of the PST to a *nonplanar* graph. According to Lipton and Tarjan [17], there exists a class of nonplanar graphs called *finite element graphs* to which their  $\mathcal{O}(\sqrt{N})$ -separator theorem still applies. The method for treating the nonplanarity problem using a finite element graph provides us some guidance for generally nonplanar graphs. A second approach would be to transform a given nonplanar graph to a planar one by adding a large number of supplemental vertices. However, with this approach the total vertex count in the augmented planar version of  $\mathbf{G}$  may be such that the “new  $\mathcal{O}(\sqrt{N})$ ” no longer offers any computational savings over competing methods [13].

An *ad hoc* procedure will be presented in Chapter 2 to partition generally nonplanar graphs. The graph partitioning approach introduced here can be used in two ways to select a set of  $\mathcal{O}(\sqrt{N})$  vertices from an  $N$ -vertex language graph. This selected set of vertices will be relatively small in number, yet will provide extremely good coverage of the paths in the graph because of the procedures used in forming the partition. It will also represent a careful selection of vertices which are well-trained. Because it is impractical to sufficiently train all the vertices (e.g., words) in a large network, it is significant that the partition selects those vertices which are well-trained. The training procedure is discussed in detail in Chapter 4.

## **1.2.2 Graph Search in the Presence of Partitioning**

### **1.2.2.1 First Pass: Modified Stack Decoding**

Two important issues related to this partitioned graph search techniques are addressed in this work. One is the ability to search with unknown boundaries in the observation sequence. The other important issue is concerned with the ability to perform the repartition of the subgraph following a given search. This second issue is only explored superficially in this research, but the method for recursive partitioning and search follows directly from the methods described here. A two-pass graph search technique is developed in this work to solve these problems. This search method integrates existing techniques to yield a large-vocabulary speaker-independent continuous-speech recognition algorithm requiring only a modest amount of computation, and with performance comparable to that of previous recognizers. A brief discussion of the two-pass graph search technique is presented below.

The unknown boundary problem is solved inherently in a modified stack search algorithm to be presented. One advantage of partitioned graph search is the reduction of the computational complexity which occurs in this search as a result of the vertex selection. In conjunction with stack search, a vertex evaluation algorithm is employed at the selected vertices to extend paths over various segments of observation strings. This procedure hypothesizes all the potential time boundaries of the selected words (by graph partitioning) in the input speech utterance. These methods are presented in Chapter 3.

To search the paths in the partitioned case, a modified left-to-right procedure

is used. The search is carried out using a modified “stack” algorithm [14], i.e., the evaluation of vertices occurs as the *selected* vertices are encountered along paths, and the possible boundaries and likelihood for each selected vertex is obtained from the results of the word-spotting. When the search encounters a vertex which is not in the set  $\mathcal{C}$ , a time duration model is used for evaluation. The time duration distributions are estimated from training data. A formal discussion is given in Chapter 3. Each evolving path is entered into a “stack” (memory array), its position in the stack is determined by its likelihood, defined as the negative logarithm of the probability of that path. The most likely partial path is put at the top of the stack.

Pruning strategies can also be applied to reduce the search space. Since the stack is of finite length, say  $q$ , only the  $q$  most likely partial paths survive. The finite stack, therefore, effects one type of pruning operation called *hard pruning* [13]. A second type of pruning occurs when a partial path, for which there is sufficient room in the stack, is deemed too unlikely to be viable and is removed. This type of pruning has been called *soft pruning* [13]. At each iteration, the partial path in the top location of the stack is extended by one word. When a complete path appears as an entry at the top of the stack, the decoding is complete. It is desirable to retain the  $n$ -best sentences. This is achieved by delaying termination until  $n$  sentences have been output. The surviving paths in the stack after the first stage of search form a subgraph, which will be significantly scaled down with respect to the original problem. Then, the second stage of the search is used to find the most likely path.

### 1.2.2.2 Second Pass: Optimal Solution

In a large problem, the subgraph remaining in the stacks after the first partition and search can be further partitioned and searched in a similar manner. The solution would be expected to rapidly converge. Each partition and search involves  $\mathcal{O}(\sqrt{N})$  or fewer evaluations.

The goal in the second stage of search is to find the most likely path in the remaining subgraph after the first stage of search. In this work, a *bigram* grammar [8], [21], is employed in conjunction with a small and well-trained graph. A bigram grammar finds the list of words that can follow any given word, and estimates the probability of a word followed by another word. Since the remaining subgraph is small, a left-to-right method (either Viterbi search or stack algorithm) can be used to search for the best path. Here we reuse the modified stack procedure employed in partition search, but each vertex is evaluated in the second pass.

---

# Chapter 2

## Algorithms for Graph Partitioning

---

### 2.1 Introduction

Vertex partitioning of graphs is applied in “divide-and-conquer” approaches [30] to solving combinatorial problems. The underlying concept in this method is to divide the original problem into independent multiple subproblems of roughly the same size. The subproblems are then solved recursively and combined to provide a solution to the larger problem.

For problems expressible as graphs, separator theorems [17], [18], [19] specify different classes of graphs to which the divide-and-conquer strategy can be applied. Here, only a vertex separator is discussed. A vertex separator is a set of vertices whose removal disconnects the graph into two subgraphs, each of which contains no more than some constant fraction of the original graph. Other definitions of separators and separator theorems are found in [22], [23]. The purpose of this chapter is to develop an algorithm for vertex partitioning.

This chapter is organized as follows. First, a detailed method for constructing a language graph (i.e., a finite-state network) is presented. Some essential graph-theoretic concepts underlying separator theorems are discussed. Algorithms for graph partitioning are then outlined, and a complete discussion of the key operation for finding a cycle for the planar graph subpartitioning is presented. Elementary notations and definitions used in this discussion are found in Appendix A.

## **2.2 Building a Language Graph**

### **2.2.1 The Need for a Large Graph**

Most speech recognition tasks can be organized into a hierarchy of finite-state networks with a finite number of vertices and edges corresponding to acoustic, phonetic, and syntactic knowledge sources and their interactions [5], [11], [14], [24]. The representation of knowledge sources in a finite-state network has been applied to both isolated-word recognition and continuous-speech recognition [11], [15], [25].

Before presenting the algorithm for vertex selection, it is necessary to describe the procedure for mapping a given speech recognition task into a finite-state network. Very large grammar graphs are preferable because they more closely represent the underlying language, and because the elemental models of the graph are trained “in context.” This is illustrated by the following simple example. Consider the language with an allowable set of messages (assumed equiprobable):

1. The father is going to walk with the child.



2. The father is going to walk the dog.
3. The child is going to walk the dog.

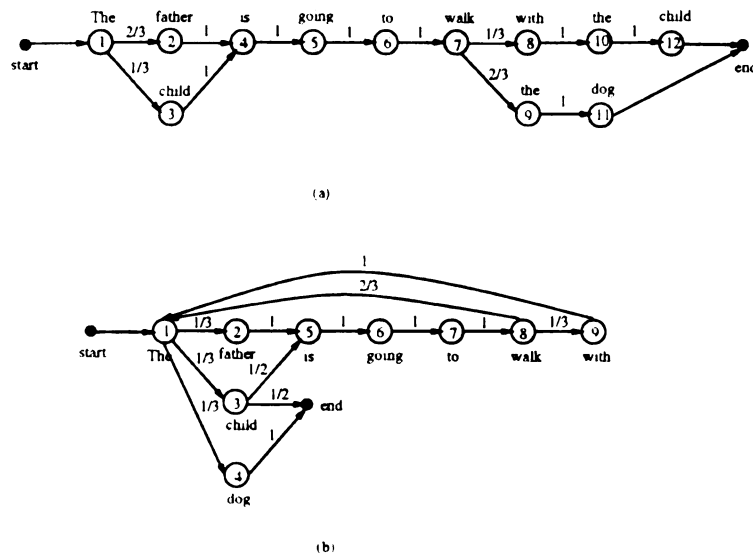


Figure 2.4: Two possible graphs which can “generate” sentences 1 – 3 listed in the text.

In Fig. 2.4 two possible graphs which can generate sentences 1 – 3 are illustrated. The larger graph in Fig. 2.4 (a) is less<sup>5</sup> “perplex.” The lower perplexity arises in the larger graph because it more accurately models *exactly* those precise messages that are in the language. To the extent that the speaker adheres to the language, there are two reasons why the larger graph is preferable: First, in the larger graph there are fewer “illegal” (not in the language) paths that the recognizer may explore (and potentially even declare to be the spoken message). From a statistical point of view, there are fewer decisions and sources of error. Secondly, because elements of spoken utterances are affected by their context within the whole utterance, it is desirable for

<sup>5</sup>The *perplexity* of a language roughly refers to the average branching factor at any decision in the graph. See [8] for a rigorous discussion.

the models for various message elements to be trained in the proper context. This will occur for the larger graph.

The smaller graph shown in Fig. 2.4 (b) has several desirable features which represent tradeoffs to the large graph benefits: First, there is added “richness” in the language in the sense that the user may utter some messages which are not in the original corpus (and have them recognized correctly). For example, the sentence “The child is going to walk with the dog” has effectively been augmented to the original language by virtue of the smaller graph. Secondly, a smaller amount of training data is needed to statistically “train” the models represented in the smaller graph simply because there are fewer of them. Thirdly, the added perplexity offers more of a challenge to any research algorithm. The lack of sufficient training data to properly train large graphs, combined with a desire to work on a challenging problem of relatively high perplexity, are the reasons why most contemporary speech recognition work has been carried out with language models represented by relatively small graphs. The partitioned search technique will permit the use of “large” language models which imply the lowered perplexity and training-in-context benefits discussed above. However, these two “large model” benefits do not come at the expense of insufficient training because the partitioning procedure presented in this chapter will select well-trained vertices.

## 2.2.2 The Method for Building the Language Graph

For the present continuous-speech recognition task, it is instructive to decompose the language graph into two levels, a grammar level and an intraword level. The two levels have completely different properties. The intraword level is usually a word model, which could be a whole-word template, a whole-word *hidden Markov model* (HMM) [8], [21], [24], or a word representation in terms of subword models such as phones<sup>6</sup>. In this work, we will focus our attention on the use of both the whole-word HMMs and a word representation in terms of phones. The reasons are described below, and a detailed discussion of implementation issues appears in Chapter 4. A complete discussion of the use of the HMM in speech recognition is found in Appendix D. An example of a six-state left-to-right model, which is also a finite-state network for representing a word, is shown in Fig. 2.5. The topology of a finite-state network for representing a phone is similar to the one in Fig. 2.5, but with fewer states and transitions.

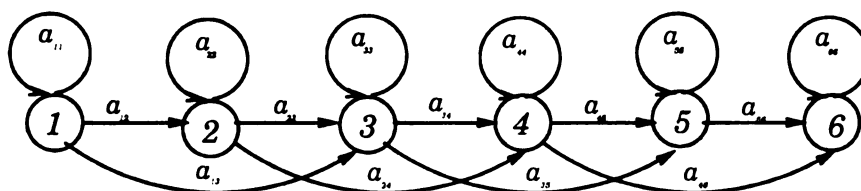


Figure 2.5: An example of a left-to-right 6-state model, where  $a_{ij}$  is the state transition probability from state  $i$  to state  $j$ .

The reasons why we select both whole-word HMMs and the word representation in terms of phone models for representing the intraword level are the following:

- Hidden Markov modeling is a powerful technique capable of robust modeling

---

<sup>6</sup>A phone is a single speech sound represented by a phonetic symbol [8].

of speech. An HMM is a parametric model that is particularly suitable and succinct for describing speech events. Moreover, it requires less storage than many other strategies [8], [21]. HMMs have two stochastic processes which enable the modeling not only of acoustic phenomena, but also of timescale distortions. Furthermore, efficient algorithms exist for accurate estimation of HMM parameters.

- Words are the most natural units of speech. Word models are able to capture within-word contextual effects. Therefore, when word models can be adequately trained, they will usually yield the best performance [21]. However, the memory usage grows linearly with the number of words since there is no sharing among words. For a large-vocabulary system, say  $10^6$  -  $10^{11}$  states [5], [11], it becomes unrealistic to train all necessary word models. In [21], for example, word-dependent phones and context-dependent phones (triphones) are found to be two appealing units since both of them are trainable for a large-vocabulary task. Therefore, the following approach is developed to represent the intraword level in the partitioned cases: A small number of the well-trained whole-word HMM's is used to represent the selected words (which are relatively small in number as discussed in Chapter 1). However, if there are selected vertices which are not represented by well-trained whole-word HMM's, then the word representations in terms of the phones are used to represent those selected words. Therefore, the training process for a large-vocabulary system in the partitioned cases is solved. More discussion of the training issues is found in Chapter 4.

The grammar level is represented by a graph in which the vertices represent the whole-word models and the edges represent word transitions<sup>7</sup>. To use this language graph for recognition, we could put the HMMs for each of the grammar vertices (i.e., words) in parallel, and allow a transition from the last state of word A to the first state of word B if (A,B) is a “legal” word pair. In a simplified model, the transition probability of this transition would be  $1/n$ , where  $n$  is the number of words that could follow word A (word-pair grammar). The bigram grammar described in Chapter 1 can also be employed. By replacing each grammar vertex on the language graph with the corresponding word models (or word representations in terms of phone models), we obtain a complete language graph showing all internal and grammar vertices and edges. An example of a partial language graph is shown in Fig. 2.6.

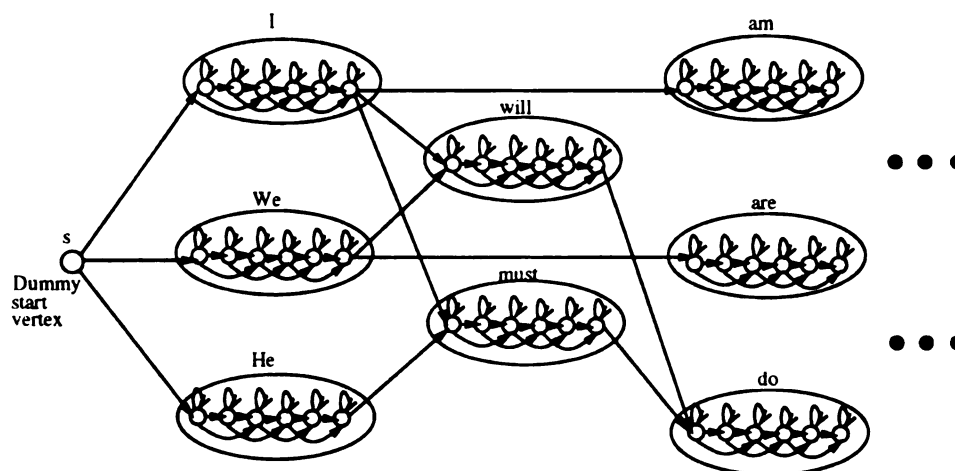


Figure 2.6: An example of a partial language graph.

The decoding task in speech recognition is the problem of associating a set of acoustic observations with a best path in the finite-state network. The appropriate

---

<sup>7</sup>Alternatively, the vertices may represent the word boundaries, and the edges represent whole-word models and word transitions.

network to search is the complete language graph as constructed above. To find the most likely word string means finding the most likely path in the graph, given the observation string and the *a priori* structure embedded in the graph. In Chapter 3, a novel “two-pass graph search” method is presented.

## 2.3 Graph Partitioning Algorithms

### 2.3.1 Partitioning Algorithms for Planar Graphs

The main focus of this subsection is on graph partitioning algorithms for planar graphs. Since the coverage issue is critical to proper pruning, we want to select an algorithm for partitioning which will provide maximal coverage. In [20], it is found that the vertices selected in the set  $\mathcal{C}$  using the algorithm based on the PST of Lipton and Tarjan (L&T) contain at least the vertices on two levels ( $l_0$  and  $l_2$  in Fig. 2.7). However, if the algorithm of Djidjev is applied, it is possible that the vertices in the set  $\mathcal{C}$  lie on only one special level as shown in Fig. 2.8. Inferior coverage of the graph to that of L&T will result. To support this argument, some experiments have been done to partition a language graph. For most graphs, it is necessary to find a cycle to complete an appropriate vertex partition when the PST of L&T is applied. The partitioning results will also give better coverage than the results created by Djidjev’s method. Therefore, we select the PST of L&T for the graph partitioning in this work.

The algorithm developed in this work for vertex partitioning is based on the solution of L&T. In turn, the constructive proof of the PST depends on two fundamental

lemmas (see Appendix B). The partitioning steps are as follows:

**Step 1** *Partition the vertices into levels according to their distance from some vertex  $s$  in the graph  $\mathbf{G}$ . Let  $L(l)$  be the total number of vertices on level  $l$ , and let  $l_1$  denote a level such that the total reward of levels 0 through  $l_1 - 1$  does not exceed  $1/2$ , but the total reward of levels 0 through  $l_1$  does exceed  $1/2$ .*

**Step 2** *Find two levels, say  $l_0$  and  $l_2$ , such that  $l_0$  is the highest level and  $l_2$  is the lowest level which satisfies the following conditions:*

$$l_0 \leq l_1 < l_1 + 1 \leq l_2$$

$$L(l_0) + 2(l_1 - l_0) \leq 2\sqrt{k}$$

$$L(l_2) + 2(l_2 - l_1 - 1) \leq 2\sqrt{N - k}$$

*where  $k$  is the number of vertices on level 0 through  $l_1$ . If the vertices on levels  $l_0$  and  $l_2$  are deleted from  $\mathbf{G}$ , then the remaining vertices of  $\mathbf{G}$  are separated into three parts (all of which may be empty): vertices on levels 0 through  $l_0 - 1$ , vertices on levels  $l_0 + 1$  through  $l_2 - 1$ , and vertices on levels  $l_2 + 1$  and above.*

**Step 3** *If the total reward on levels  $l_0 + 1$  through  $l_2 - 1$  does not exceed  $2/3$ , let  $\mathbf{C}$  be the set of vertices on levels  $l_0$  and  $l_2$ , let  $\mathbf{A}$  be the part of the three with most reward, and let  $\mathbf{B}$  be the remaining two parts. Then, the vertex partitioning is complete. The algorithm is terminated.*

**Step 4** *Otherwise, shrink all the vertices on levels 0 through  $l_0$  to a single vertex  $x$*

of reward zero and delete all the vertices on levels  $l_2$  and above to form a new graph  $G'$ . Note that  $G'$  is planar [17]. Apply the cycle-finding algorithm presented in the next section to find a cycle of the new graph  $G'$  to complete the vertex partitioning. Then, the vertices in set  $C$  are the vertices on levels  $l_0$  and  $l_2$  plus the vertices on the cycle found in the new graph  $G'$ . The vertex partitioning is complete.

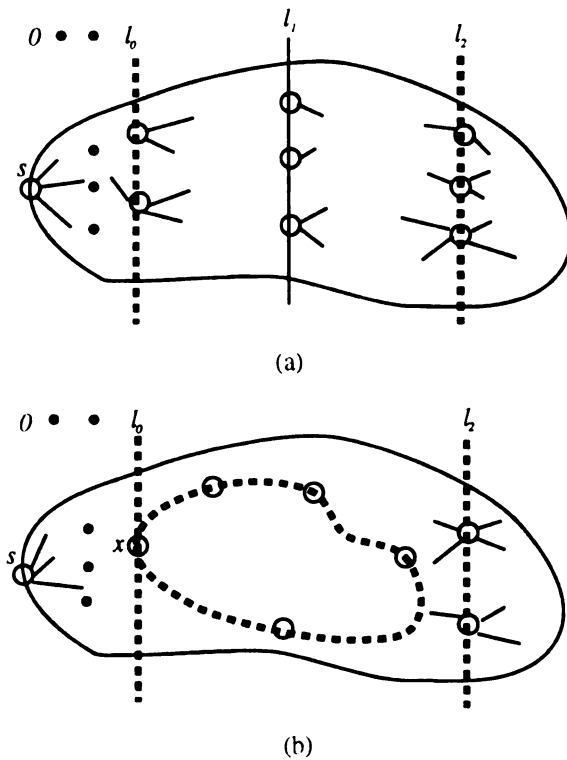


Figure 2.7: (a) A conceptual plot of the location of the vertices in the set  $C$  for case (1) in the proof of the PST of Lipton-Tarjan. The set  $C$  is represented by the dotted lines in the plot; (b) The location of the vertices in the set  $C$  for case (2).

### 2.3.2 Partitioning Nonplanar Graphs

An *ad hoc* procedure to deal with nonplanarity has been developed in this research. Suppose that the PST of L&T is used. Let  $G$  be a nonplanar graph, and let  $G''$  be a planar graph formed from  $G$  such that all edges causing nonplanarity in  $G$  are



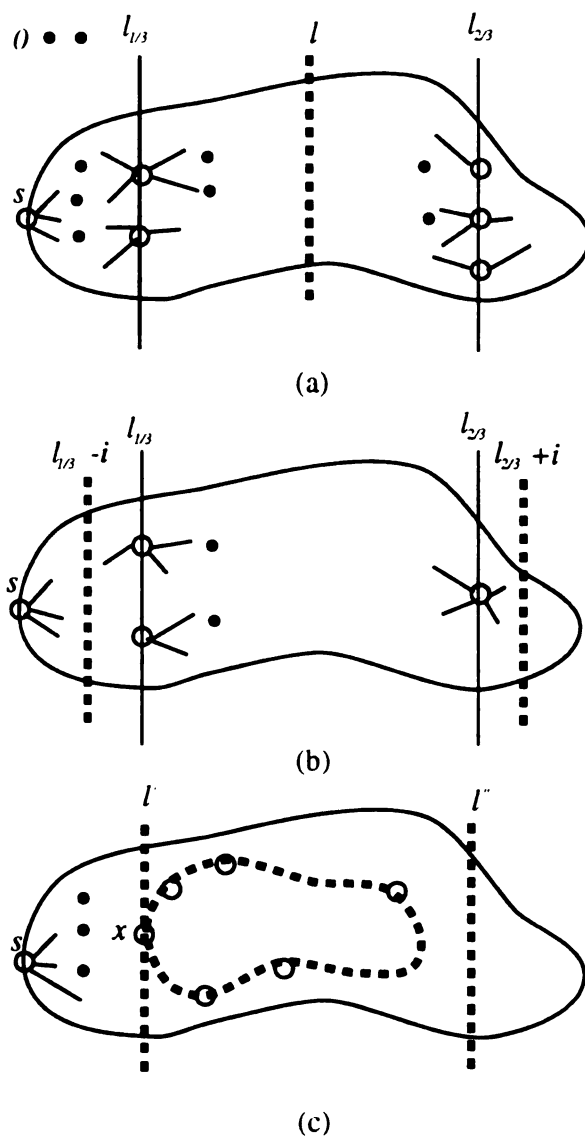


Figure 2.8: (a) A conceptual plot of the location of the vertices in the set  $C$  for case (1) in Djidjev's version of the PST. The set  $C$  is represented by the dotted lines in the plot; (b) The location of the vertices in the set  $C$  for case (2a). (c) The location of the vertices in the set  $C$  for case (2b).

extracted and set aside for future reference. Then  $G$  and  $G^*$  have the same number of vertices but different numbers of edges.  $G^*$  is said to be a planar subgraph of  $G$ . Assume that the PST of L&T is applied to  $G^*$ . Let  $A^*$ ,  $B^*$  and  $C^*$  be the resulting partition. Now, form  $G$  from  $G^*$  by bringing back the edges causing nonplanarity according to the following procedures: If certain planarity-breaking edges do not join  $A^*$  and  $B^*$ , then they can be restored without changing the partitioning result in  $G^*$ . That is, we may assign  $C$  to be all the vertices in  $C^*$ . For planarity-breaking edges which violate the PST (e.g., some planarity-breaking edges join  $A^*$  and  $B^*$ ), then assign one end vertex of the planarity-breaking edge to the  $C$  set (originally, it might be in the  $A^*$  or  $B^*$  set). Therefore, the violation is eliminated. Let  $A$  be the remaining vertices in  $A^*$ , and let  $B$  be the remaining vertices in  $B^*$ . Then no edge in  $G$  joins  $A$  and  $B$ , neither  $A$  nor  $B$  has total reward exceeding  $2/3$ .

By inspection, there are at most three cases which will cause violation of the PST and which arise from the planarity-breaking edges being returned to the graph. These cases are shown in Fig. 2.9. Since the vertex partitioning is not unique, one can always choose a good approximation of the  $\mathcal{O}(\sqrt{N})$ -separator for the nonplanar case by inspecting the violation of the PST when it is applied to nonplanar graphs.

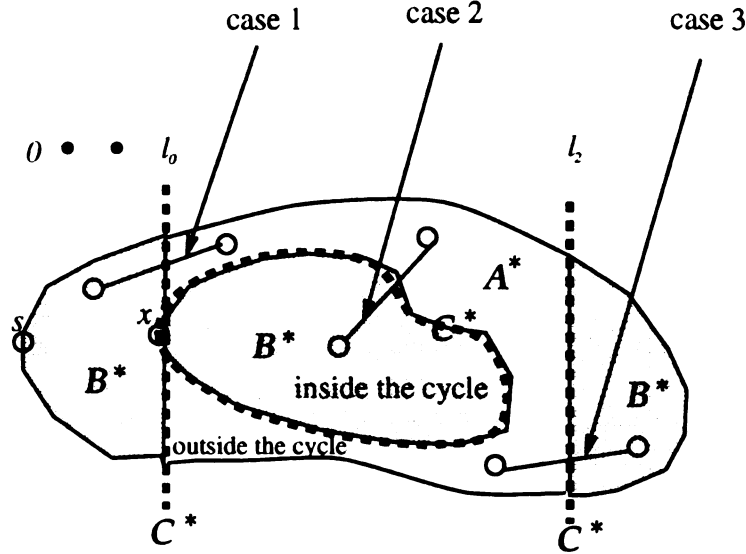


Figure 2.9: Three cases which will cause the violation of the planar separator theorem when the edges causing nonplanarity are restored.

## 2.4 Cycle-Finding Algorithm

### 2.4.1 Edge-Scanning Approach

Algorithms for finding a vertex partitioning satisfying the PST have been outlined in the previous section. An important issue not discussed there is how to find a subpartitioning of the new graph  $G'$  formed by the shrinking and deleting operations. Lipton and Tarjan have presented an algorithm for finding the cycle to complete the vertex partitioning (see Appendix C). However, the operation in Step 3 of their approach is difficult to implement since the computer cannot distinguish which tree edges incident to the cycle should be located to the inside and which should be to the outside. Therefore, the “edge-scanning” approach, developed as part of this research [20], is used to find a proper cycle to complete the vertex partitioning. The edge-scanning approach is a significantly improved version of the one described in [16],

which was developed in the earlier stages of this research.

The edge-scanning approach simplifies the procedures for finding a cycle in vertex subpartitioning. The method has three key features which distinguish it from the L&T approach:

1. There is no need to construct a new representation for the planar embedding. Consequently, the proposed method is good for partitioning in real time.
2. There is no need to triangulate all faces of the new graph, thereby eliminating the need for adding new edges to this new graph in advance.
3. The algorithm uses a simple rule to determine which vertices should be considered inside of the cycle and which vertices outside.

As a consequence of these simplified strategies, the new cycle-finding algorithm is easy to implement.

The block diagram for the edge-scanning approach is presented in Fig. 2.10. The algorithm consists of four main modules: (1) a planarity testing (PT) module to run the Demoucron algorithm (see Appendix D) [1] which is used to determine the boundary of each face in the planar embedding of  $\mathbf{G}'$ , say  $\mathbf{H}'$ ; (2) a breadth-first search (BFS) module to find a breadth-first spanning tree of  $\mathbf{H}'$ ; (3) an original nontree edge (ONE) module to determine whether there exists a nontree edge, originally in this newly formed graph  $\mathbf{H}'$ , which forms a cycle that satisfies Lemma 2 in [17]; (4) an added nontree edge (ANE) module. If there is no nontree edge in  $\mathbf{H}'$  which forms a desired cycle in the ONE module, by the proof of L&T one can find a desired cycle by

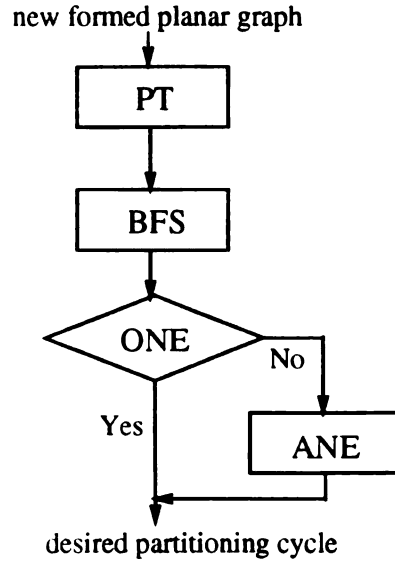


Figure 2.10: The block diagram for finding the partitioning cycle in the new graph.

adding new edges to  $\mathbf{H}'$ . This module provides a systematic way to add a new edge which forms a desired cycle to complete the vertex partitioning. Modules PT and BFS are essentially the inputs of this cycle-finding algorithm. We will now describe modules ONE and ANE in more detail.

The block diagram of the ONE module is shown in Fig. 2.11. Choose any nontree edge from  $\mathbf{H}'$ , say  $(v, w)$ , and form the corresponding cycle using  $(v, w)$  and some tree edges in the graph  $\mathbf{H}'$ . If neither the inside nor the outside of the chosen cycle has total reward exceeding  $2/3$  of the reward in the graph  $\mathbf{G}'$ , then the cycle is the desired one in the planar graph subpartitioning of  $\mathbf{G}'$ . A problem arises here. How can it be determined which vertices should be considered inside of the cycle and which vertices outside? A cycle-finding algorithm (CFA) is presented to solve this problem.

Suppose one nontree edge  $(v, w)$  is chosen in  $\mathbf{H}'$ . Let the two incident faces be  $f'$  and  $f''$ . For convenience, let  $F$  be the set of the faces in  $\mathbf{H}'$ , let  $F_o$  be the set of the

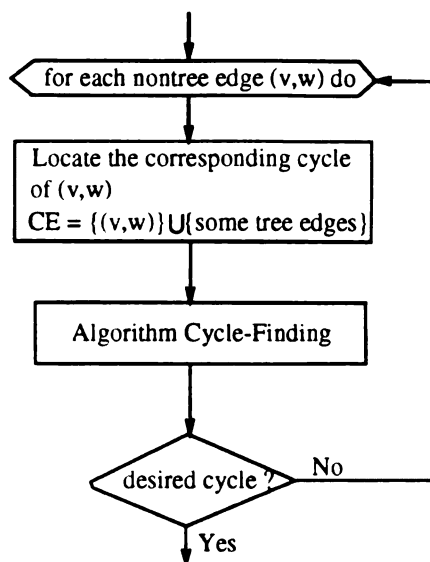


Figure 2.11: The block diagram for the ONE module.

faces in  $\mathbf{H}'$  to the outside of the chosen cycle, and let  $F_i$  be the set of the faces in  $\mathbf{H}'$  to the inside of the chosen cycle. Once a fixed planar embedding of  $\mathbf{G}'$ , viz.  $\mathbf{H}'$ , is found, it is simple to obtain a list of all the faces to which each edge in  $\mathbf{H}'$  belongs by scanning the boundaries of the faces. This list must be prepared before the CFA is invoked. The input to the CFA can be either  $f'$  or  $f''$ . Without loss of generality,  $f'$  can be chosen to be the input to the CFA, and the face is assigned to be located to the inside of the chosen cycle. The algorithm for determining which face is located to the inside of the cycle and which face is located to the outside is as follows:

### Cycle-Finding Algorithm( $\mathbf{H}'$ )

- 1 mark all the faces and edges in  $\mathbf{H}'$  "unused"
- 2 mark all the edges on the chosen cycle "used"
- 3  $F_i \leftarrow \{f'\}$
- 4 mark  $f'$  "used"

- 5  $f_{input} \leftarrow f'$
- 6 **Determination** ( $f_{input}$ )

The algorithm **Determination** is as follows:

**Determination** ( $f_{input}$ )

- 1 **for** each edge  $e \in f_{input}$  satisfying:
  - (i)  $e$  is an “unused” edge
  - (ii)  $e$  has more than one “unused” face incident to it (Check the list of all the faces to which the edge  $e$  belongs.)
- do**
- 2 mark edge  $e$  “used”
- 3 **for** each face, say  $\tilde{f}$ , incident to  $e$  and  $\tilde{f}$  is an “unused” face
  - do**
  - 4 mark face  $\tilde{f}$  “used”
  - 5  $F_i \leftarrow F_i \cup \{\tilde{f}\}$
  - 6  $f_{input} \leftarrow \tilde{f}$
  - 7 **Determination** ( $f_{input}$ )

The faces in  $F_i$  are the faces in  $\mathbf{H}'$  to be located to the inside of the chosen cycle. Thus, the vertices which should be located to the inside of the chosen cycle are all the vertices in set  $F_i$ , excluding the vertices on the chosen cycle. Once the vertices to the inside of the chosen cycle are determined, the vertices to the outside of the cycle

are known. The proof of correctness of the CFA is straightforward. Since the number of the faces located to the inside of the chosen cycle is finite, the CFA will therefore terminate in a finite number of steps.

If, on the other hand, there does not exist a set of nontree edges in the graph  $G'$  that forms an appropriate cycle as described in the ONE module, then the ANE module is applied. The block diagram for ANE module is shown in Fig. 2.12. In this module, one new edge is added in a face for each test. The new edge in the face is denoted as a *chord* of this chosen face, i.e., the new edge is an edge  $(v, w)$  such that  $v$  and  $w$  are nonadjacent vertices on the boundary of the face. The newly added edge is a nontree edge which can form a cycle with some tree edges. A new edge added to a face will divide the original face into two parts, each part forming a new face of the planar embedding. Each time a new edge is added to a face, it is determined whether the corresponding cycle formed by the new added edge satisfies the condition that neither the inside nor the outside of the cycle has reward exceeding  $2/3$  of the reward in the graph  $G'$ . The ONE module is applied for this determination. Therefore, the proper cycle can be found.

### 2.4.2 An Example

The method for finding a cycle is illustrated by an example. The original graph  $G$  is shown in Fig. 2.13. Without loss of generality, all vertices in  $G$  are assigned equal reward values which sum to unity. For visual understanding of the graph, this graph is embedded in the plane as shown in Fig. 2.14 using Demoucron's planarity



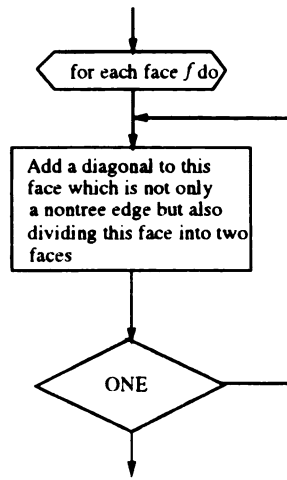


Figure 2.12: The block diagram for the ANE module.

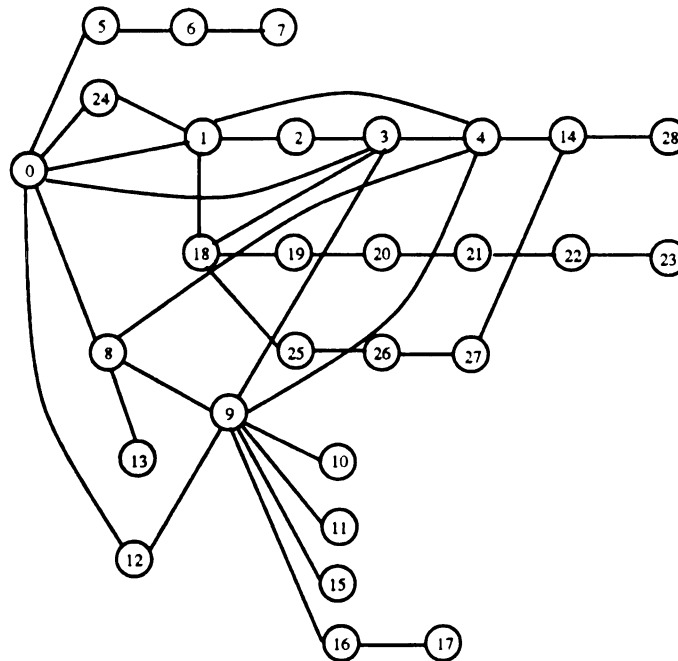


Figure 2.13: An example graph used to illustrate the subpartitioning algorithm.

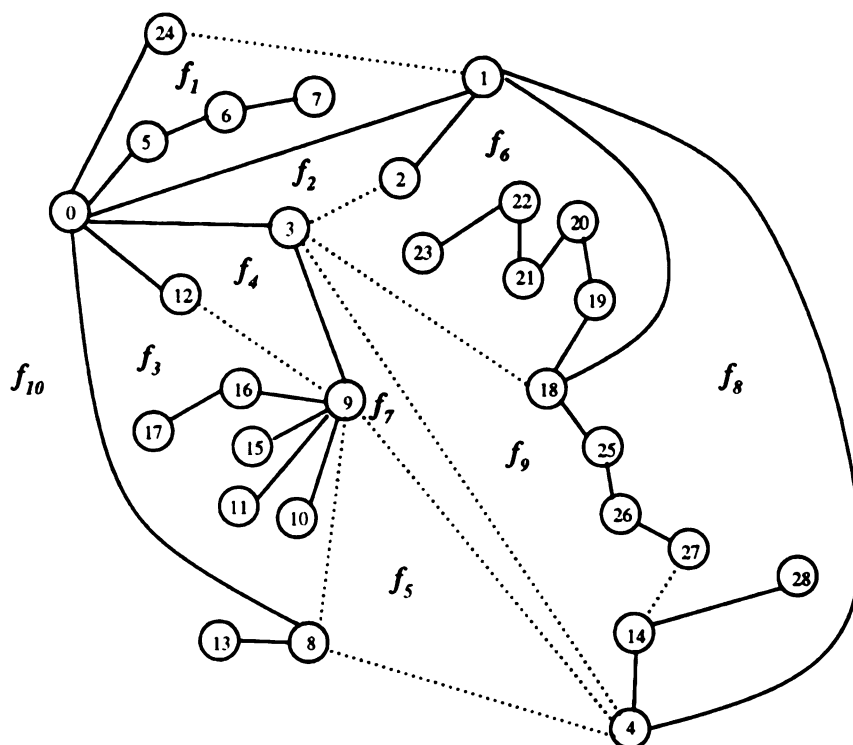


Figure 2.14: The planar embedding of the planar graph of Fig. 2.13, where the solid lines represent the tree edges and the dotted lines are the nontree edges.

algorithm [1]. The set of vertices in each face of the planar embedding is listed in Table 2.1. The identified breadth-first spanning tree is represented by the solid lines in Fig. 2.14. The nontree edges are shown by dotted lines. The vertices on each level are shown in Table 2.2. If the PST of L&T is applied to this example, it is necessary to find a cycle in the new graph  $G'$  (formed by the shrinking and deleting processes) to complete the vertex partitioning. By using Demoucron's planarity algorithm, the boundary of each face is found, and the new graph is embedded in the plane. The planar embedding of  $G'$ , say  $H'$ , is shown in Fig. 2.15 and the set of vertices in each face of  $H'$  is listed in Table 2.3.

The breadth-first spanning tree is also shown in Fig. 2.15. After using the tech-

Face	Vertices
face 1	0, 1, 5, 6, 7, 24
face 2	0, 1, 2, 3
face 3	0, 8, 9, 10, 11, 12, 13, 15, 16, 17
face 4	0, 3, 9, 12
face 5	4, 8, 9
face 6	1, 2, 3, 18, 19, 20, 21, 22, 23
face 7	3, 4, 9
face 8	1, 4, 14, 18, 25, 26, 27, 28
face 9	3, 4, 14, 18, 25, 26, 27
face 10	0, 1, 4, 8, 13, 24

Table 2.1: The set of vertices in each face of the planar embedding of the graph in Fig. 2.13.

Level	Vertices
Level 0	0
Level 1	1, 3, 5, 8, 12, 24
Level 2	2, 4, 6, 9, 13, 18
Level 3	7, 10, 11, 14, 15, 16, 19, 25
Level 4	17, 20, 26, 27, 28
Level 5	21
Level 6	22
Level 7	23

Table 2.2: The vertices on each level of the graph in Fig. 2.13 after running BFS.

Face	Vertices
face 1	0, 1, 5, 6, 7, 24
face 2	1, 2, 3, 18, 19, 25
face 3	0, 8, 9, 10, 11, 12, 15, 16
face 4	0, 3, 9, 12
face 5	4, 8, 9
face 6	1, 2, 3, 4
face 7	3, 4, 9
face 8	0, 1, 3, 18
face 9	0, 1, 4, 8, 13, 14, 24

Table 2.3: The set of vertices in each face of the planar embedding of the newly formed graph.

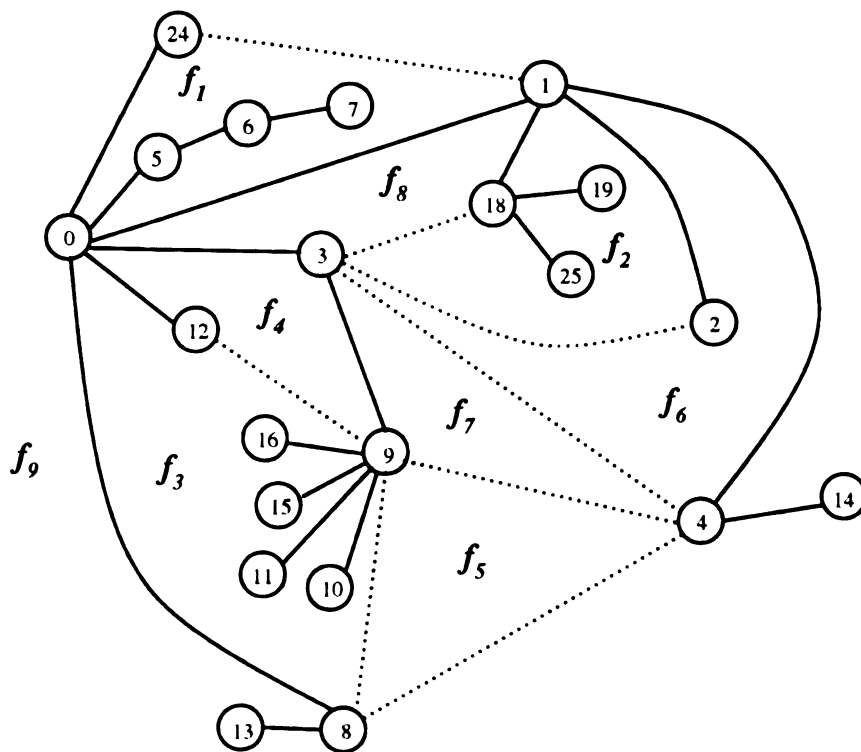


Figure 2.15: The planar embedding of the newly formed graph in this example, where the solid lines represent the tree edges and the dotted lines show nontree edges.

Face	Vertices
face 1	5, 6, 7, 24
face 2	2, 18, 19, 25
face 3	8, 9, 10, 12, 15, 16
face 4	9, 12
face 5	8, 9
face 6	2
face 7	9
face 8	18
face 9	8, 13, 14, 24

Table 2.4: The set of vertices in each face of  $\mathbf{H}'$  except the vertices which are on the chosen cycle.

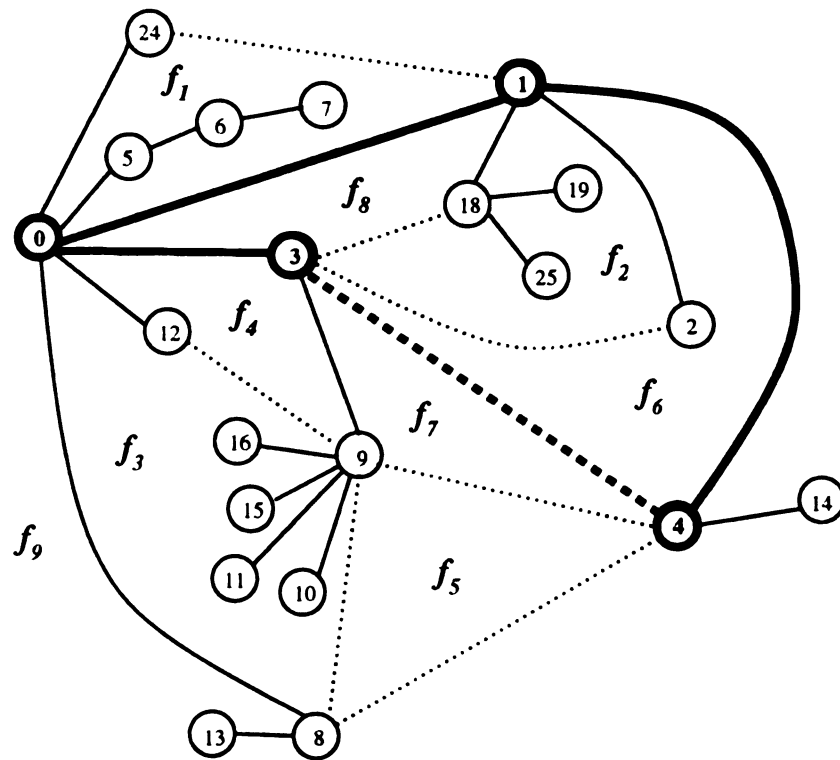


Figure 2.16: A desired cycle, which is shown by the bold lines in the figure, is found by the cycle-finding algorithm. Note that the dotted edge (3,4) is a nontree edge.

nique presented in this section, one nontree edge,  $(3,4)$ , and its corresponding cycle are found. The edges of the cycle are  $(0,1), (1,4), (4,3), (3,0)$ . This chosen cycle is shown in Fig. 2.16. The set of vertices in each face, except the vertices which are on the chosen cycle, are listed in Table 2.4. The vertices inside and outside the cycle are separated as described below:

All the faces of  $\mathbf{H}'$  are marked “unused.” The edges, on the chosen cycle are marked “used” and the rest marked “unused.” Note that an edge cannot be common to more than two faces, and in this case since it happens to be on the cycle, one of the faces subtended must be inside and the other outside the cycle. Either one of these faces is tagged “inside” and the other “outside.” The vertices associated with an inside or outside face are also necessarily inside or outside the cycle respectively, and they are tagged accordingly. In the example,  $f_6$  and  $f_7$  are two incident faces to the nontree edge  $(3,4)$ . Face  $f_6$  is marked “used.” Let  $f_6$  be the input to the algorithm **Determination**.

Starting with an unused edge of face  $f_6$ , a search is conducted for an unused face incident on it. Since the edge already bounds an inside face, the other unused face incident on it must also be inside the cycle. Once found, the edge and the incident face are both marked “used.” The vertices bounding this newly found face will then be inside vertices. In Fig. 2.16, the unused edge  $(2,3)$  of  $\mathbf{H}'$  is found to be common to faces  $f_6$  and  $f_2$ . Face  $f_2$  and edge  $(2,3)$  are marked “used.”

The process above is carried out exhaustively for all the inside faces. The faces determined to be located inside the cycle are  $f_2, f_6$ , and  $f_8$ . Finally, it can be determined which vertices are located on the inside of the cycle, and which vertices

	Vertices	Total Number
Inside	2, 18, 19, 25	4 ( $< 2/3 N$ )
Outside	5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 24	13 ( $< 2/3 N$ )

Table 2.5: The set of vertices on each side of the chosen cycle.

outside the cycle. These are recorded in Table 2.5. The vertices in the set  $C$  are the vertices on the chosen cycle plus the vertices on the two levels. For this example, these two levels are level 0 and level 4. There are a total of nine vertices on the set  $C$ . These are vertices 0, 1, 3, 4, 17, 20, 26, 27, and 28. A visual understanding of the location of sets  $A$ ,  $B$ , and  $C$  after partitioning is obtained by viewing Fig. 2.17. One should notice that the result of the vertex partitioning is not unique [17].

If, after the subpartitioning, the total number of vertices inside and outside the cycle in  $H'$  does not satisfy Lemma A.2, another cycle can be chosen and the process repeated. If none of the existing cycles is found suitable, a particular face can be triangulated and a new cycle chosen. The graph could be incrementally triangulated until an appropriate partitioning cycle is found. From Fig. 2.15, it is evident that further trials are not necessary for this example.

---

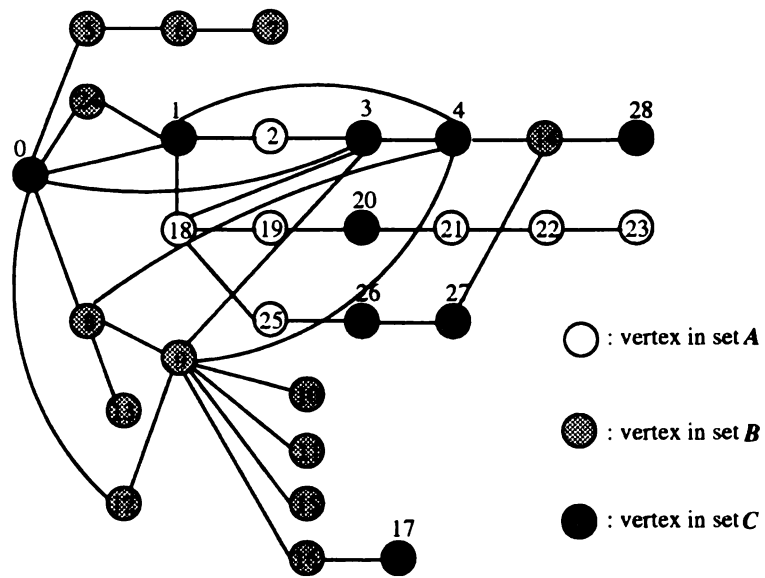


Figure 2.17: The visual understanding of the partitioning results of the example graph.



# Chapter 3

## A Stack Algorithm for Partitioned Search

---

### 3.1 The $n$ -Best Strategy

The partitioned graph search techniques presented here are based on a type of “ $n$ -best” philosophy which has become popular in recent years [57]. The general  $n$ -best search paradigm is as follows: Apply the most powerful and inexpensive knowledge sources first to produce a scored list of the top  $n$  paths. Then, this list of the top  $n$  paths is reordered using the remaining (“expensive”) knowledge sources to arrive at the most likely solution. Fig. 3.18 illustrates such an  $n$ -best search paradigm.

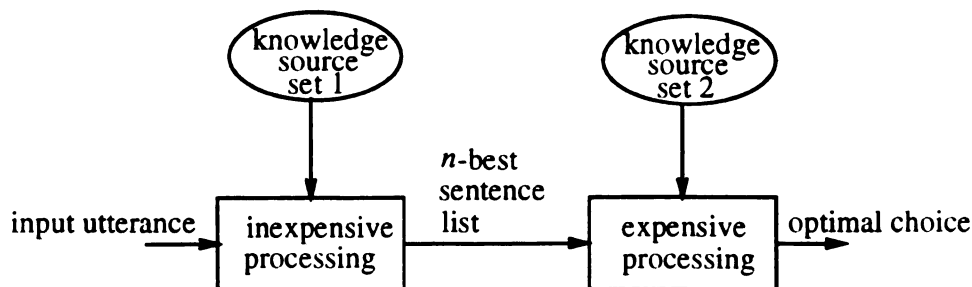


Figure 3.18: The  $n$ -best search paradigm.

In this research, the partitioning theorem described in previous chapter is applied to continuous-speech recognition whose high-level constraints are expressible as a finite-state network. The application of the theorem drastically reduces the computational complexity involved in the search of such graphs. Off-line graph partitioning is used to locate a relatively small number of key vertices which cover a significant number of paths. The graph is searched and pruned according to a likelihood measure, with some subset of these selected vertices evaluated en route. This search under “scattered” evaluation requires a modified decoding strategy to be described in this chapter. At the end of the search, the problem is then pared down to one involving a smaller graph of likely solutions (i.e., the top  $n$  surviving paths after the graph is searched and pruned). The remaining graph can be further partitioned and searched in a similar manner if it is undesirably large. Then, the smaller graph is subjected to intense scrutiny to find an optimal solution.

In the following sections, a two-pass graph search technique is presented to conduct the search in the partitioned cases. The first stage of the search uses a modified stack decoder in conjunction with a vertex evaluation subroutine to efficiently reduce the graph to a subgraph consisting of the “ $n$ ” best paths. This small graph comprises a small search space of usually much less than size  $\mathcal{O}(\sqrt{N})$ , whereas the original graph is of size  $N$ . The second stage of the search seeks out the optimal path from among those remaining in the subgraph. At the end of the search, the most likely path corresponding to the input utterance is found.

## 3.2 First Pass: Modified Stack Decoding

### 3.2.1 Vertex Evaluation Algorithm

#### 3.2.1.1 Baseline Algorithm

Because it is based on conventional principles, we first describe the procedure by which a selected vertex is evaluated when encountered during the search process. It will be noted that this procedure is akin to conventional word-spotting [73], but is carried out in the context of a top-down search [8] so that the benefits of the grammar may be exploited.

In the following discussion, the evaluation technique is presented for a the case in which a whole-word model is present at the vertex. However, for those selected vertices represented by phone models, a similar technique can be applied. The only difference is that the whole-word model is replaced by combining the relevant phone models into a word HMM. In this case, if the probability of the between-phone transition is set to  $\alpha$ , then the last state self-transition probability (prior to the phone transition) is changed to  $1 - \alpha$  (no longer being 1) if there exists a between-phone transition. Figure 3.19 illustrates how to combine the phone models into a word HMM.

Suppose it is desired to evaluate the model  $M^p$  with respect to the partial observation sequence  $\mathbf{Y}_{t',t''} = y_{t'}, \dots, y_{t''}$  for any times  $t' \leq t''$ . Let  $s_t$  denote the state at time  $t$ . Let  $\delta_{t'}^{t''}(k)$  denote the joint probability of the partial observation sequence  $\mathbf{Y}_{t',t''}$ , and arriving at state  $k$  at time  $t''$ , i.e.,  $s_{t''} = k$ , given a specific whole-word

HMM, say  $\mathbf{M}^p$ ,

$$\delta_{t''}^{t''}(k) = P(\mathbf{Y}_{t',t''} \text{ and } s_{t''} = k | \mathbf{M}^p). \quad (3.1)$$

Also let  $blike(t_{start}, t)$  be the likelihood (negative log of the probability) of associating the partial observation sequence  $\mathbf{Y}_{t',t''}$  with the *best* sequence of states (Viterbi decoding [6]) in the model,

$$blike(t', t'') \stackrel{\text{def}}{=} -\log P(\mathbf{Y}_{t',t''} \text{ and } s_{t'}^*, \dots, s_{t''}^* | \mathbf{M}^p) \quad (3.2)$$

where  $s_{t'}^*, \dots, s_{t''}^*$  indicates the optimal sequence of states. Suppose we wish to evaluate the  $l^{th}$  vertex on a path,  $x_l$ , which contains model  $\mathbf{M}^p$ . The observation string  $\mathbf{Y}_{1,t_1-1}$  has been associated with the vertices on the path leading up to  $x_l$ , so the observation string for  $\mathbf{M}^p$  must begin at time  $t \geq t_1$ . We allow for some tolerance in the choice of starting time,

$$t_1 \leq t_{start} \leq t_1' \stackrel{\text{def}}{=} \min\{t_1 + \bar{D}_l, T\} \quad (3.3)$$

where  $\bar{D}_l$  is the average length of all words in the training data appearing in the  $l^{th}$  position in an utterance, and  $T$  is the total number of observations in the string being decoded. In seeking endpoints for the evaluation, we allow

$$t_{start} + 1 \leq t_{stop} \leq t_2 \stackrel{\text{def}}{=} \min\{t_1 + \bar{D}_p + \sigma_p, T\} \quad (3.4)$$

where  $\bar{D}_p$  and  $\sigma_p$  are the mean and standard deviation of the training sequence dura-

tions for model  $\mathbf{M}^p$ . Then, the basic evaluation then follows the frame-synchronous Viterbi algorithm presented below (see Appendix E for details).

### Vertex Evaluation Algorithm

```

    {Initialization}

1  for each  $t_1 \leq i \leq t'_1$ 
    do

2       $\delta_i^i(1) = b_1^p(y_i)$ ,
3       $\delta_i^i(j) = 0$ ,  $2 \leq j \leq I = \text{total states in } \mathbf{M}^p$ 
4      if  $\delta_i^i(1) < \Delta_l$ , next  $i$ .

    {Recursion}

5      for each  $i + 1 \leq t \leq t_2$ , and  $1 \leq j \leq I$ ,
    do

6           $\delta_i^t(j) = \max\{\delta_i^{t-1}(k) a_{kj}^p b_j^p(y_t) : 1 \leq k \leq I\}$ ,

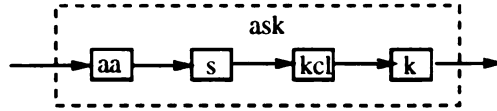
    {Termination}

7      if  $\text{blike}(i, t) < \Delta_u$  and  $(t - i + 1) > \Delta_L$ , then
8          record  $\delta_i^t(I)$ ,  $i + 1 \leq t \leq T$ .

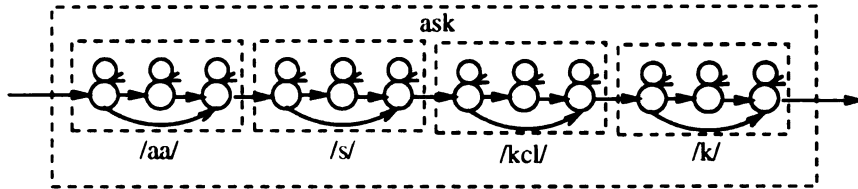
```

The notations  $a_{ij}^p$  and  $b_j^p(y_t)$  indicate standard HMM probabilities which are defined in Appendix E. Three thresholds,  $\Delta_L$ ,  $\Delta_l$  and  $\Delta_u$ , are set in this algorithm to find acceptable intervals.  $\Delta_L$  is the minimum length of a candidate interval.  $\Delta_l$  is the lower-bound probability for any potential interval which starts at time  $i$ , i.e., if  $\delta_i^i(1) > \Delta_l$ , then time  $i$  is a possible starting time of a potential interval in the

observation sequence corresponding to the given word model.  $\Delta_u$  is the upper-bound likelihood for any promising interval, any interval with likelihood exceeding  $\Delta_u$  is excluded. Briefly, the vertex evaluation algorithm presented above works as follows: For every possible starting time  $i$ , i.e.,  $\delta_i^i(1) > \Delta_l$ , find the largest ending time  $t$  such that  $blike(i, t) < \Delta_u$  and  $(t - i + 1) > \Delta_L$ .



(a)



(b)

Figure 3.19: An example illustrates how the phone models can be combined to form a big word model.

### 3.2.1.2 Refining Boundaries Following Vertex Evaluation

After executing the baseline evaluation algorithm above, many potential intervals for the extension of the path over  $x_l$  may be hypothesized. The likelihood for each surviving interval will usually be very close to  $\Delta_u$ . However, one can still tell which interval is most likely by inspecting the lengths of these potential intervals. The best result is one which has the smallest likelihood per observation time. However, there is a problem: There will generally be many overlapping potential intervals detected by the baseline algorithm. Therefore, the space for storing these results is large. A

merging criterion presented below is used to build a stack which contains a relatively small number of entries. At the same time, a reasonable likelihood measure for the merged boundaries is described. Finally, the merged potential boundaries and their corresponding likelihoods are stored in the stack.

In designing a merging criterion and likelihood measure, two requirements are considered. One is that those intervals with good likelihoods are retained after they are merged. The other is that the new likelihood resulting from the merger of boundaries must be reasonable, which means the merger does not decrease the likelihood to an unacceptable level (the length of the merged interval is usually increased).

### A. Merging Criterion

Given two hypothesized word intervals  $B^1$  and  $B^2$  from the results of the baseline algorithm, let  $B^i_{start}$  and  $B^i_{stop}$  be the starting time and ending time for the hypothesized interval  $B^i$ , respectively. Then, there are three possible cases for the relationships among the boundaries of  $B^1$  and  $B^2$ :

1. One interval is contained in the other, say  $B^2 \subseteq B^1$ . Then, the merged interval  $B$  is the same as  $B^1$ . This case is shown in Fig. 3.20 (a).

2. There is no intersection between  $B^1$  and  $B^2$ , e.g.,  $B^1_{stop} < B^2_{start}$  or  $B^2_{stop} < B^1_{start}$ .

In this case, no merging occurs. This case is shown in Fig. 3.20 (b).

3.  $B^1$  and  $B^2$  partially intersect. Assume  $B^2_{start} \leq B^1_{stop} < B^2_{stop}$ . If  $(B^2_{stop} - B^1_{stop}) < \Delta_L$ , then the boundaries  $B^1$  and  $B^2$  can be merged as one interval  $B$  such that  $B_{start} = B^1_{start}$  and  $B_{stop} = B^2_{stop}$ . Otherwise, no merging occurs between  $B^1$

and  $B^2$  because the boundary gap between them is large enough to retain both as hypothetical intervals. This case is shown in Fig. 3.20 (c).

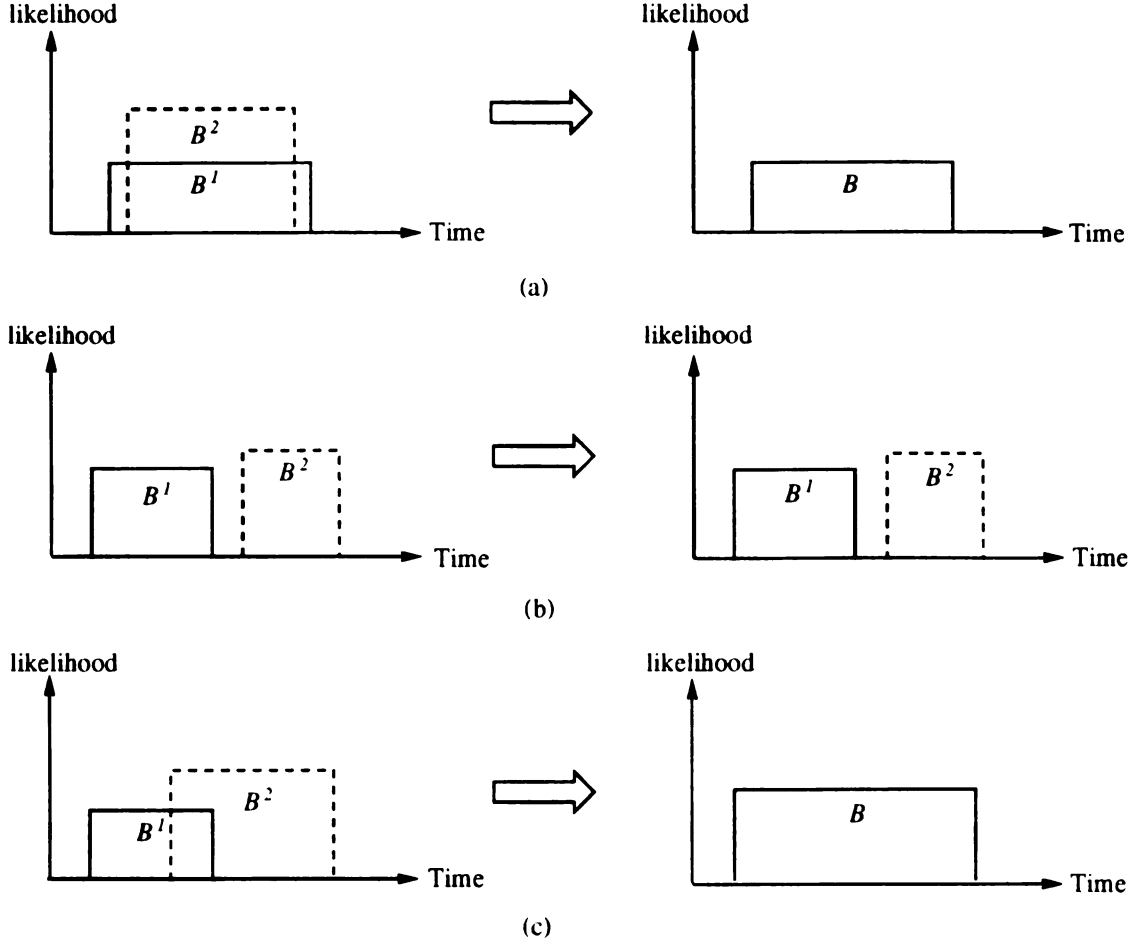


Figure 3.20: Three possible cases of merging potential intervals  $B^1$  and  $B^2$ , which are the resulting potential intervals from the baseline word spotting algorithm: (a)  $B^2 \subseteq B^1$ ; (b) no intersection between  $B^1$  and  $B^2$ ; (c)  $B^1$  and  $B^2$  are partially intersected.

## B. Likelihood Measure for Merged Boundaries

Suppose the intervals  $B^1, B^2, \dots, B^k$  result from the merging process above. Then, the following likelihood measure is used to calculate the likelihood of these merged boundaries. Let  $like(B^i_{start}, B^i_{stop})$  be the likelihood of the resulting interval  $B^i$ . Then, the likelihood is of the form:



$$like(B_{start}^i, B_{stop}^i) = blike(B_{start}^i, B_{stop}^i) + (\text{duration likelihood}) \quad (3.5)$$

where the duration likelihood is computed as follows: Since matching the word model  $\mathbf{M}^p$  to some part of the observation sequence is essentially unconstrained, it is possible to expand or contract the subsequence assigned to a model so that it accounts for a large or a small part of the observation sequence. This problem is alleviated by adding the duration constraints to likelihood measure. A simple Gaussian duration model is used. Let  $P_p(D_i)$  be the probability density of the potential interval  $B^i$  with length  $D_i$  for the word model  $\mathbf{M}^p$ . Then:

$$P_p(D_i) = \frac{1}{\sqrt{2\pi}\sigma_p} e^{-[\frac{(D_i - \bar{D}_p)^2}{2\sigma_p^2}]} \quad (3.6)$$

where the mean and standard deviation for  $\mathbf{M}^p$  are  $\bar{D}_p$  and  $\sigma_p$ , respectively. These statistics are estimated from the training data. Then, the “duration likelihood” is just the negative logarithm of  $P_p(D_i)$ . The conceptual plot of the word spotting results for the selected vertices (i.e., words) is shown in Fig. 3.21.

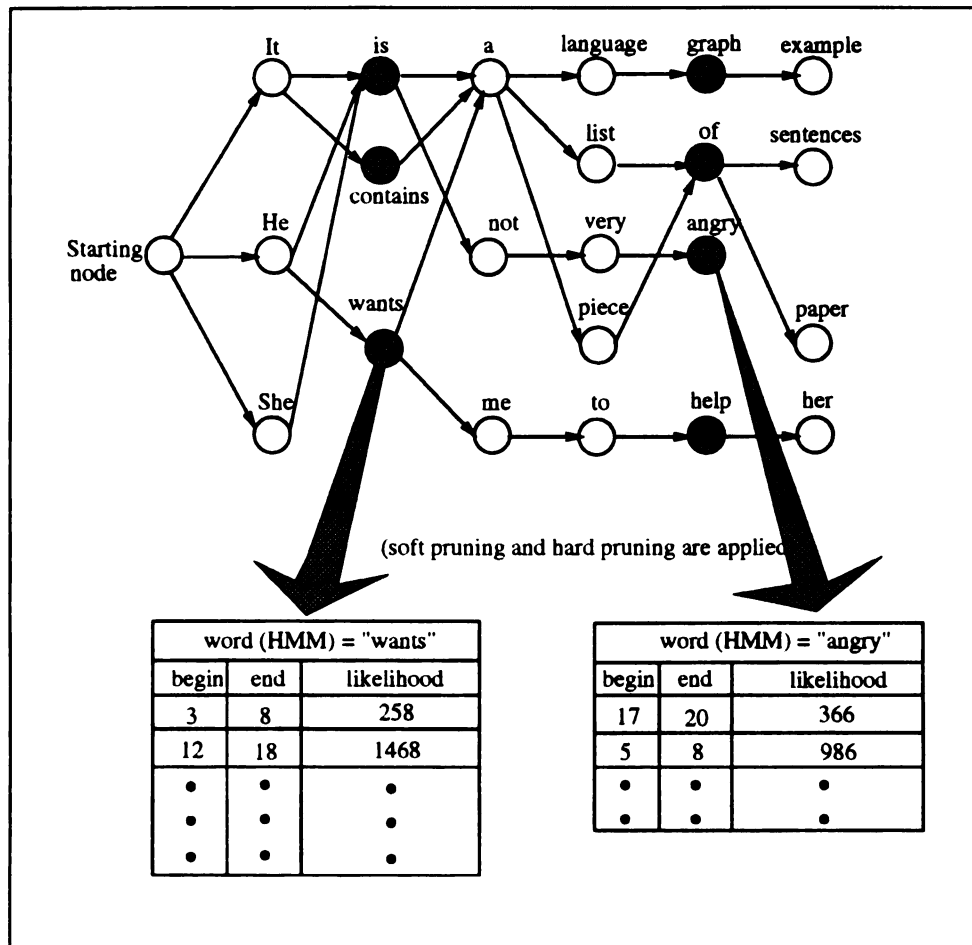


Figure 3.21: A conceptual plot of the word-spotting results for the selected vertices. The vertices represented by shaded circles are vertices selected through the graph partitioning approach.

### 3.2.2 Modified Stack Decoding

In conventional *path driven* left-to-right search strategies, the evaluation of vertices takes place as they are encountered along paths. In partitioned case, where the objective is to eliminate this costly procedure by preselecting vertices for evaluation. Here, a modified left-to-right procedure is presented. This method involves some rather straightforward modifications of conventional stack decoding procedures [11], [58] to accommodate the unevaluated vertices.

The essence of the partitioning method is that only  $\mathcal{O}(\sqrt{N})$  vertices in  $\mathbf{G}$  are selected for evaluation. This means that an alternative method must be available to “evaluate” an unselected (not on the selected set  $\mathbf{C}$ ) vertex. A simple and conservative procedure is used to replace the evaluation model by a time-duration model at each unselected vertex extension. When the quantity of the probability of an unselected vertex extension is needed, we use, instead, the probability that the observation string corresponding to this unselected vertex extension is a given length. These probabilities are estimated from the training data as described above.

For unselected vertices, a simple Gaussian duration model is used, i.e., the probability density for an unselected vertex, say  $x_p$ , is of the form (3.6) where  $x_p$  and  $\mathbf{M}^p$  are equivalent in these two discussions since model  $\mathbf{M}^p$  is resident at vertex  $x_p$ . Thus, the duration model is incorporated into the partitioned graph search techniques. Accordingly, a modified stack decoding algorithm for the partitioned graph search is as follows:

## Modified Stack Decoding Algorithm

**{Initialization}**

Put the start vertex, say  $s$ , in the stack to form a “null” partial path.

**{Recursion: Best-First Path Growing Loop}**

Take the top entry (the best partial path) off the stack, say  $\mathbf{X}_{0,l-1} = s, x_1, \dots, x_{l-1}$ .

If the best partial path is complete (i.e., the end-of-path flag is “true”). Then,

- i. Output the path and increment the output hypothesis counter by one.
- ii. If the output hypothesis counter is equal to  $n$ , stop.

Else

For each possible vertex extension, say  $x_l$ , of the partial path

(i.e.,  $B_{start}^{l-1} < B_{start}^l$ )

**do**

- i. If  $x_l$  is an unselected vertex, then a time-duration model is used.  
to update the likelihood (see below)
- ii. If  $x_l$  is a selected vertex, then either a whole-word HMM or  
a phonetic-baseform model is used (see below).
- iii. The pruning strategy discussed below is applied.

It is clear that the stack itself is just a sorted list which supports the operations: take the best entry off the stack and insert new entries according to their likelihoods. The following must be contained in each stack entry:

1. The history of the partial path.

2. The partial path likelihood.
3. An end-of-path flag to indicate whether the entry is a complete path.
4. The beginning time in the observation sequence corresponding to the terminal vertex of the partial path.

Each path extension is entered into the stack, its position determined by its likelihood. The stack is of finite length, say  $q$ , so that only the  $q$  most likely partial paths survive. The finite stack, therefore, effects the pruning operation called hard pruning in Chapter 1. A second type of pruning occurs when a partial path, for which there is sufficient room in the stack, is deemed too unlikely to be viable and is removed. This process is called soft pruning. The best  $n$  paths are determined through the following process: At each iteration, paths are extended from the top of the stack down. The decoding is complete when  $n$  complete paths appear as entries in the top  $n$  entries of the stack.

Likelihood updates during the extensions occur according to the following arguments: The method described below is based on the report by Venkatesh, *et al.* [13] although differences exist in the present method from the one described there. Suppose the likelihood of a particular path through the graph  $\mathbf{G}$  represented by the vertex string  $\mathbf{X} = x_1, x_2, \dots, x_L$  is evaluated, and assume that the likelihood measure is the probability of the occurrence of path  $\mathbf{X}$ , given the observation string  $\mathbf{Y} = y_1, y_2, \dots, y_T$ . Using Bayes rule,  $P(\mathbf{X} | \mathbf{Y}) = P(\mathbf{X}, \mathbf{Y})/P(\mathbf{Y})$ . Since  $P(\mathbf{Y})$  is

identical for all paths, it is sufficient to seek the path  $\mathbf{X}^*$  such that

$$\mathbf{X}^* = \arg \max_{\mathbf{X}} P(\mathbf{X}, \mathbf{Y}). \quad (3.7)$$

Let us assume that the best *partial* path in the stack ranges over the vertices  $\mathbf{X}_{1,l-1} = s, x_1, \dots, x_{l-1}$ , and that the best likelihood associates observations  $\mathbf{Y}_{1,t_1-1}$  with this partial path, i.e.,  $t_1 - 1$  maximizes  $-\log P(\mathbf{X}_{1,l-1}, \mathbf{Y}_{1,\tau})$  over all  $\tau$ . Then we wish to compute the likelihood associated with extending the path over vertex  $x_l$  using observations  $\mathbf{Y}_{t_1,t}$  where  $t \geq t_1$ . Note that

$$\begin{aligned} P(\mathbf{X}_{1,l-1}, \mathbf{Y}_{1,t_1-1} \cap x_l, \mathbf{Y}_{t_1,t}) &= P(\mathbf{X}_{1,l-1}, \mathbf{Y}_{1,t_1-1}) P(x_l, \mathbf{Y}_{t_1,t} \mid \mathbf{X}_{1,l-1}, \mathbf{Y}_{1,t_1-1}) \quad (3.8) \\ &= P(\mathbf{X}_{1,l-1}, \mathbf{Y}_{1,t_1-1}) P(x_l \mid \mathbf{X}_{1,l-1}, \mathbf{Y}_{1,t}) P(\mathbf{Y}_{t_1,t} \mid x_l, \mathbf{X}_{1,l-1}, \mathbf{Y}_{1,t_1-1}) \\ &= P(\mathbf{X}_{1,l-1}, \mathbf{Y}_{1,t_1-1}) P(x_l \mid x_{l-1}) P(\mathbf{Y}_{t_1,t} \mid x_l). \end{aligned}$$

We have made extensive use of the assumed independence of the observations in this derivation. We see that the likelihood extension is simple. The quantity  $P(x_l \mid x_{l-1})$  is the transition probability associated with the graph edge between  $x_{l-1}$  and  $x_l$ , and  $P(\mathbf{Y}_{t_1,t} \mid x_l)$  is obtained from the vertex evaluation procedure described above, or, if  $x_l$  is not selected for evaluation, by inserting the likelihood that the duration of the sequence generated by the model at  $x_l$  is  $t - t_1$ . This latter number is obtained from the duration density for the model.

When the best  $n$  paths have been obtained, the question remains as to how to select the optimal path. What remains in the stack are surviving paths which represent

a small subgraph of  $G$ , say  $G'$ . Depending on the scale of the problem, it is useful to “partition”  $G'$  using another set of vertices and repeat the procedure<sup>8</sup>, or simply to search  $G'$  using the standard left-to-right method or some other *ad hoc* technique. Whatever the case, the new search problem will be very significantly scaled down with respect to the original problem. Using these methods, the search will converge to a solution very quickly. A grammatical approach is presented in next section to find the most likely path after the first-pass search. Note that the repartition of  $G'$  cannot be done in real time in the current research due to the limitations of the machine speed and memory space. Many other interesting search procedures may arise with future research.

### 3.2.3 Computation Savings from the First Pass

The evaluation process comprises a vast majority of the computational effort of the search algorithm above. In fact, each attempt to extend a path by one vertex involves multiple evaluations of the vertex with respect to about  $T(T + 1)/2$  separate substrings of the observation sequence,  $Y$ , where  $T$  is the length of the complete observation sequence. In the case in which a HMM is present at the vertex, each of these “subevaluations” will require between  $\mathcal{O}(HT')$ , and  $\mathcal{O}(H^2T')$  operations (depending on the model structure), where  $H$  is the number of states in the HMM and  $T'$  denotes the substring length<sup>9</sup>. The other computations required in the search become

---

<sup>8</sup>It is not necessary to “start over,” but rather to supplement the information in the existing stack with further evaluations.

<sup>9</sup>Further improvement might be possible using an HMM evaluation method due to Deller and Snider [12].

insignificant in this light, and the computational expense is seen to be nearly directly proportional to the number of vertices actually evaluated. The partitioning approach, therefore, can be seen to reduce this cost by a factor which is  $\mathcal{O}(\sqrt{N})$  with respect to a full search of  $\mathbf{G}$ .

It is difficult to quantify the benefit of the partitioning approach in terms of the cost a conventional left-to-right search, since the later is highly problem-dependent. For illustrative purposes, however, consider a language graph of  $N = 10^8$  vertices, each representing a phoneme (e.g., see [11]). If there are typically 30 phonemes in a sentence, then, on the average, we would expect to find about  $33 \times 10^5$  vertices in a time slot. To evaluate only the first three time slots ( $\approx 1$  word) would require about  $10^7$  vertex evaluations. Such an evaluation is 1000 times more expensive than the  $10^4$  vertex evaluations required in the partitioned case, and does not offer the pruning safety of using acoustical data which are distributed across the message in time. Not explicitly mentioned above is the very important fact that the partitioned search is more robust to impulsive or intermittent noise than conventional left-to-right search. The evaluation of the data, by focusing on the partitioned  $\mathbf{C}$ -set of vertices, is distributed across the observations in time. This means that the search is less likely to be pruned in the presence of unmodeled noise. Also, the inherent advantage in this work is that there need be no special form of the underlying model for a symbol at a vertex, as long as some suitable measure of likelihood is computable for each vertex extension.



### 3.3 Second Pass: Optimal Solution

It is important to keep in mind that the objective of the partitioned graph search is not necessarily to deduce a single best path through a language graph. Partitioned search provides a systematic way to divide and conquer the original large graph using a series of low complexity operations. In a large problem, the subgraph remaining in the stack after the first partition and search can be further partitioned and searched in a similar manner. The solution would be expected to rapidly converge. The  $i^{\text{th}}$  partitioning recursion would require  $\mathcal{O}(N^{1/2})$  evaluations.

The goal in the second stage of the search is to find the most likely path in the remaining subgraph after the first stage of the search. In this work, we use the simple procedure of iterating over the existing subgraph with its inherent bigram grammar using the stack decoding algorithm described above. The number of resulting hypotheses,  $n$ , is set to unity (“1-best” search), and *all nodes are evaluated*. Since the graph is very small, the computational effort involved in this procedure is not significant. In future applications, any number of procedures may be used in this “expensive” pass through the graph.

# Chapter 4

## Implementation and Experimental Issues

---

### 4.1 The TIMIT Database

To evaluate the performance of the partitioned graph search techniques, the TIMIT (Texas Instruments - MIT) speech database [29] is used. The TIMIT database has been designed to provide speech data for the acquisition of acoustic-phonetic knowledge and for the development and evaluation of automatic speech recognition systems. TIMIT contains a total of 6,300 sentences, 10 sentences spoken by each of 630 speakers from eight major dialect regions of the United States. Seventy percent of the speakers are male, and most speakers are Caucasian adults. There are of total 2,342 distinct sentences, which consist of two dialect “shibboleth” sentences (sa), 450 phonetically-compact sentences (sx), and 1890 phonetically-diverse sentences (si). The database is divided into dialect regions: New England (dr1), Northern (dr2), North Midland (dr3), South Midland (dr4), Southern (dr5), New York City (dr6), Western (dr7), and Army Brat (moved around) (dr8). The “sa” sentences are spoken across all speakers, they introduce an unfair bias for certain phonemes in certain contexts [21]. Therefore,

the two “sa” sentences are not used in the current experiments. The vocabulary size in this work is 6,100. Some relevant information about the use of TIMIT in this work is summarized in Table 4.6.

	Training set	Test set
<b>No. of speakers</b>	462	168
<b>Dialect coverage</b>	8	8
<b>No. of male</b>	326	112
<b>No. of female</b>	136	56
<b>Sentence coverage</b>	sx 1- 450 si 1-1890	sx 1-450 si 1-1890
<b>No. of sentences</b>	3,696	1,344
<b>Vocabulary size</b>	4,891	2,373
<b>No. of words</b>	11,087	4,621

Table 4.6: Summary of the usage of the TIMIT database in this work.

The TIMIT corpus includes time-aligned orthographic transcriptions, phonetic transcriptions, word transcriptions as well as speech waveform data for each sentence-utterance. The file structure for the TIMIT database is shown in Fig. 4.22.

The TIMIT waveforms are sampled at 16 kHz, and the samples are 16-bit short integers which are in VAX/Intel byte order (least significant byte/most significant byte). Thus, it is required to swap the bytes in the samples to MSB/LSB order before they are used. Since the current experiments are performed on SUN workstations, the following command is used to skip the 1024-byte ASCII header and byte-swap the samples:

```
dd if=input_file.wav bs=1024 skip=1 | dd conv=swab > output_file
```

TIMIT employs a set of 62 phonetic labels. These labels, along with examples, are listed in Table 4.7.

Phone	Example	Phone	Example	Phone	Example
/iy/	bEEt	/r/	Ray	/b/	Bob
/ih/	bIt	/w/	Way	/d/	Dad
/eh/	bEt	/y/	Yacht	/g/	Gag
/ey/	bAIIt	/hh/	Hay	/p/	Pop
/æ/	bAt	/hv/	aHead	/t/	Tot
/aa/	bOtt	/el/	bottLE	/k/	Kick
/aw/	bOUt	/s/	Sea	/dx/	dirTy
/ay/	bItE	/sh/	She	/q/	baT
/ah/	bUt	/z/	Zone	/jh/	Judge
/ao/	bOUght	/zh/	meaSure	/ch/	CHurch
/oy/	bOY	/f/	Fin	/bcl/	(b closure)
/ow/	bOAAt	/th/	THin	/dcl/	(d closure)
/uh/	bOOK	/v/	Van	/gcl/	(g closure)
/uw/	bOOt	/dh/	THen	/pcl/	(p closure)
/ux/	bEAUty	/m/	MoM	/tcl/	(t closure)
/er/	bIRd	/n/	NooN	/kcl/	(k closure)
/ax/	About	/ng/	siNG	/epi/	(epin. clos.)
/ix/	deblt	/em/	bottoM	/h#/	(beg. sil)
/axr/	buttER	/en/	buttoN	/#h/	(end sil)
/ax-h/	sUspect	/eng/	washINGton	/pau/	(betw. sil)
/l/	Lay	/nx/	wiNNer		

Table 4.7: Phonetic labels used in the TIMIT database.

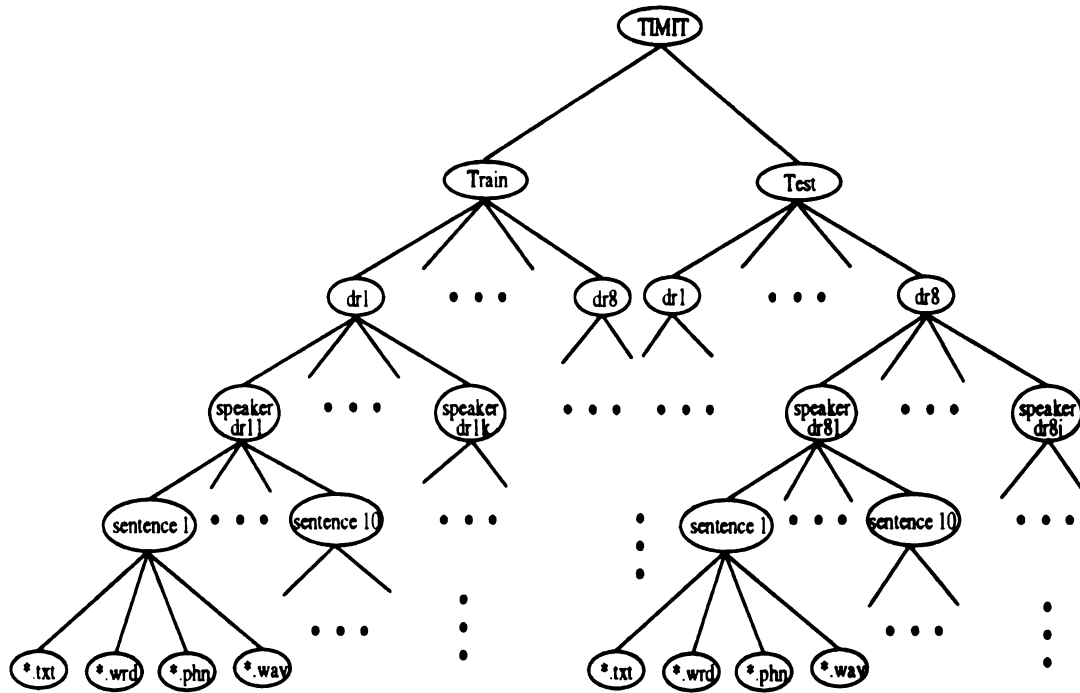


Figure 4.22: The datafile structure of the TIMIT database.

The TIMIT database is used in two ways in the experiments: First, a graph with 14,259 vertices (i.e., words) is constructed from the database. The procedures for building the graph are described in detail in next section. Second, TIMIT is used for both training and recognition. In the training process, 1,156 words, each of which has a predetermined amount of training data, are selected to train the whole-word models. Also, 60 phone models are trained for use in phonetic baseforms. In the recognition process, a test sentence from TIMIT is chosen as an input utterance, and then the partitioned graph search techniques described in Chapter 3 are applied to find the most likely path in the graph. Note that the training process is done before the graph partitioning process in this work, and both processes are off-line.

## 4.2 Implementation Issues

### 4.2.1 Building a Language Graph

The large graph for testing the partitioned graph search methods is constructed as follows: All the “sx” and “si” sentences in TIMIT database are used to build the graph. Let  $\Sigma = \{\Sigma_i\}$  be the set of sentences, which is composed of discrete words from the set,  $\mathbf{W} = \{w_i\}$ . Note that two words, say  $w_1$  and  $w_2$ , are *distinct* in the graph (have different models at different vertices) if the phonetic labels for  $w_1$  are different from that for  $w_2$ . This is true even if  $w_1$  and  $w_2$  are the same word (same orthographic transcription). For example, as illustrated in Fig. 4.23, there are eight different vertices for representing the word *ask* in the large graph. However, there is only one way to represent the word *ask* as shown in Fig. 4.24 in the bigram graph.

The  $i^{\text{th}}$  sentence (of length  $L$ ) is of the form,  $\Sigma_i = w_{i1}, w_{i2}, \dots, w_{iL}$ , where the comma denotes concatenation. The word set and the sentence length are finite, so that the set of sentences can be represented as a directed graph (digraph), say  $\mathbf{G}(\mathbf{V}, \mathbf{A})$ , in which each vertex is associated with only one word, each edge (or dart in the digraph) represents the transition from word to word in sentences, and each path represents a legal sentence. Formally speaking, the *Markov language graph*  $\mathbf{G}(\mathbf{V}, \mathbf{A})$  consists of a set of vertices,  $\mathbf{V} = \{x_i\}$ , a set of edges,  $\mathbf{A} = \{a_{kj}\}$  ( $a_{kj}$  connects vertex  $x_k$  to vertex  $x_j$ ), a special vertex  $s \in \mathbf{V}$  indicating the start vertex of each path, and a set of transition weights,  $\{P(x_j | x_i)\}$ .  $P(x_j | x_i)$  is the probability that  $w(x_i)$  is followed by  $w(x_j)$  in any sentence, where  $w(x_k)$  denotes the word associated with vertex  $x_k$ . Moreover, the elements in the set of complete paths through  $\mathbf{G}$  (which

means that those paths in  $\mathbf{G}$  begin at  $s$  and terminate at some  $x_k$ ) will have one-to-one correspondence to the elements in  $\Sigma$ . Resident at the vertex in the selected set (e.g., the set  $\mathbf{C}$  after graph partitioning) is either a whole-word HMM or a phonetic-baseform model composed of context-independent phone HMMs. Note that whole-word models are often found to outperform phone-based models if sufficient data are available to train them [8]. Whether a whole-word model or a phonetic model is used at a particular vertex depends on whether there are sufficient training data for the whole word. In addition to the HMMs at the vertices of the selected set, a simple Gaussian time-duration distribution described in Chapter 3 for the word is resident at the vertex. Note that the statistics for these distributions are tied across all occurrences of the word in the training data regardless of phonetic baseforms.

An example language graph is shown in Fig. 4.25(a). This graph is created from the following list of sentences, each sentence begins with a dummy point, say “•”, the dummy point is the start vertex  $s$  of each path in the graph  $\mathbf{G}$ . As mentioned in Chapter 3, this dummy vertex represents a silent region at the beginning of each sentence. The sentences are :

- She is thinner than I am (sx5)
- They often go out in the evening (sx187)
- I honor my mom (sx231)
- Never happier in my life (si548)
- Nor is she a wet boat (si778)

The task is to find the planar subgraph of the language graph by extracting out the planarity breaking arcs (arcs which make the language graph  $\mathbf{G}$  nonplanar) and

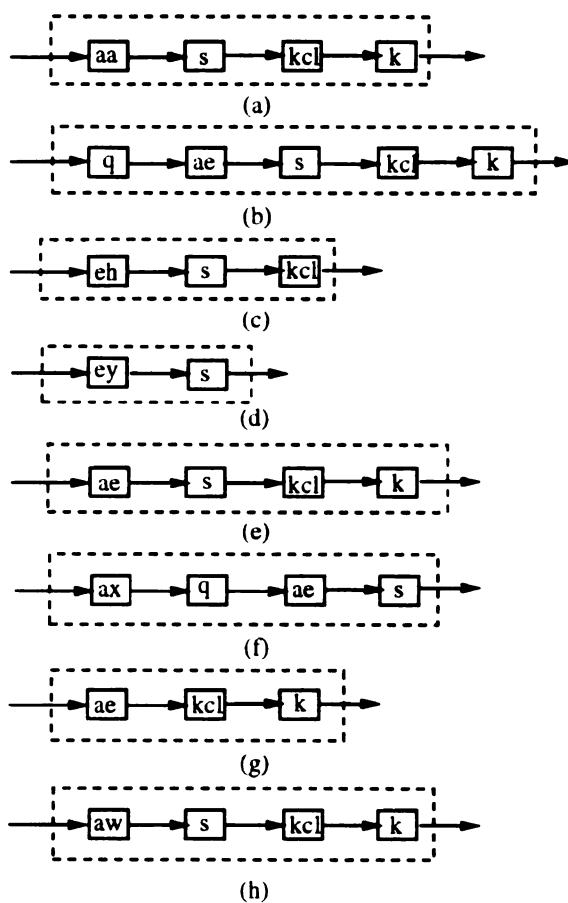


Figure 4.23: All the possible vertices with different phonetic labels for the word *ask* in a large graph which is similar to that of Fig. 2.1 (a). Each case represents a vertex in the large graph.

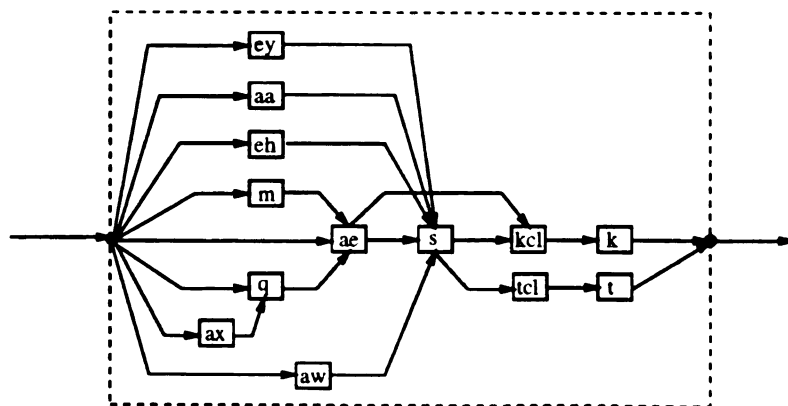


Figure 4.24: The word *ask* in a small graph which is similar to that of Fig. 2.1 (b).



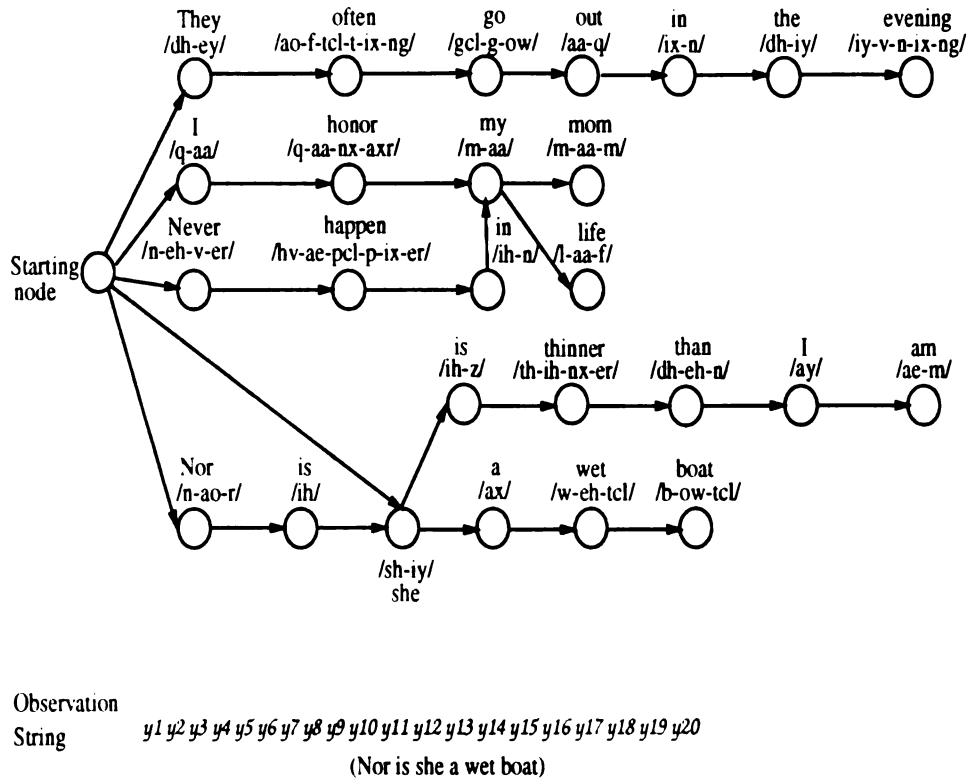


Figure 4.25: (a) An example language graph. (b) The boundaries of the observation string are unknown.

store the planarity breaking arcs for future reference. Note that the planar subgraph, say  $G'$ , must include all the vertices in the original graph  $G$ . On the other hand, let  $G(V, E)$  be the underlying undirected graph of  $G(V, A)$ , then the undirected version of  $G(V, A)$  is the undirected graph formed by converting each edge of  $G(V, A)$  to an undirected edge and removing duplicate edges. The way to construct the graph  $G$  is as follows:

A data file to store the set of sentences  $\Sigma$  is created (recall that each sentence begins with a dummy point  $\bullet$ ). The sentences are stored in the data file one by one, and every word together with its phonetic transcriptions in a sentence are stored line by line in the data file according to the concatenation of the words in the sentence. After building up a data file using the method described above, the data file is read line by line from the top. Whenever a new line is read, one new vertex may be added to the partial graph (which has been built up to this point) if the phonetic transcription in this line is different from any one in the existing partial graph. At the same time, a new edge to the partial graph is added (recall that each edge represents the transition from word to word in sentences). Since the language graph is constructed in this fashion, a *connected* directed graph is obtained. However, the underlying undirected graph might not be planar. The graph is generally too large to use the method presented in [16] for extracting the planarity breaking arcs in the graph. The following method is developed to extract the planarity breaking arcs and find the partition sets of the graph. The detailed process of the graph partitioning is described later in this chapter.

Since the existence of the start vertex increases the possibility of nonplanarity,

we can temporarily remove the start vertex from the graph. However, all the edges incident to the start vertex can be brought back in the partitioning process. The reason why we can remove the start vertex first is that: According to the search algorithms developed in Chapter 3, the start vertex is always in the selected set  $\mathcal{C}$ , and it is the first vertex to be evaluated on each path. Let us recall the partitioning algorithm for nonplanar graphs developed in Chapter 2. Any planarity breaking arc incident to a vertex in the set  $\mathcal{C}$  can be brought back without violating the PST. Therefore, we can remove the start vertex first and discover the planarity breaking arcs for each *component* of the remaining graph. Note that the remaining subgraph might not be connected. Therefore, it might contain some connected components. If some component remains large, we can further discover the *blocks*<sup>10</sup> in that component. A linear algorithm to identify the blocks of a graph is found in [4]. Since the size of each block in each component of the remaining subgraph is relatively small, we can then apply the method described in [16] to find the planarity breaking arcs for each component of the remaining subgraph. After this process, the partitioning procedure can be implemented.

## 4.2.2 Speech Modeling for Partitioned Graph Search Techniques

### 4.2.3.1 Cepstral Analysis

The feature extraction task for this speech recognition system is based on the mel-

---

<sup>10</sup>A block is a graph which contains no cut vertex.

cepstral parameters. The analysis in this research is based on a paper by Davis and Mermelstein [27], in which the mel-based cepstral coefficients, say  $\mathbf{c}_n$ , are calculated as below:

$$\mathbf{c}_n = \sum_{i=1}^{20} \mathbf{E}_i \cos\left[n\left(i - \frac{1}{2}\right)\frac{\pi}{20}\right] \quad \text{for } n = 1, 2, \dots, K \quad (4.9)$$

where  $\mathbf{E}_i$  denotes the *critical band*<sup>11</sup> filter log energy outputs, and  $K$  is set to 8 in our implementation. Twenty mel-frequency components are desired on the Nyquist range 0 – 8 kHz (the sampling rate is 16 kHz). The critical-band filtering is then simulated by a set of twenty triangular bandpass filters as shown in Fig. 4.26, where ten mel-frequency components appear linearly on the Range 0 – 1 kHz and the remaining ten are distributed logarithmically on the range 1 – 8 kHz.

#### 4.2.3.2 Vector Quantization

One major issue in vector quantization (VQ) is the design of an appropriate codebook for quantization. The VQ codebook contains a set of  $L$  vectors which provide the minimum average distortion (distance) between a given training set of analysis vectors and the codebook entries. The most common measure of difference between two vectors, say  $\mathbf{c}_1$  and  $\mathbf{c}_2$ , is the Euclidean distance  $d(\mathbf{c}_1, \mathbf{c}_2)$  (suppose that there are  $K$  features in each vector):

$$d(\mathbf{c}_1, \mathbf{c}_2) = \sqrt{\sum_{i=1}^K [\mathbf{c}_1(i) - \mathbf{c}_2(i)]^2}. \quad (4.10)$$

---

<sup>11</sup>A critical band can be viewed as a bandpass filter whose frequency response corresponds roughly to the tuning curves of auditory neurons [76].

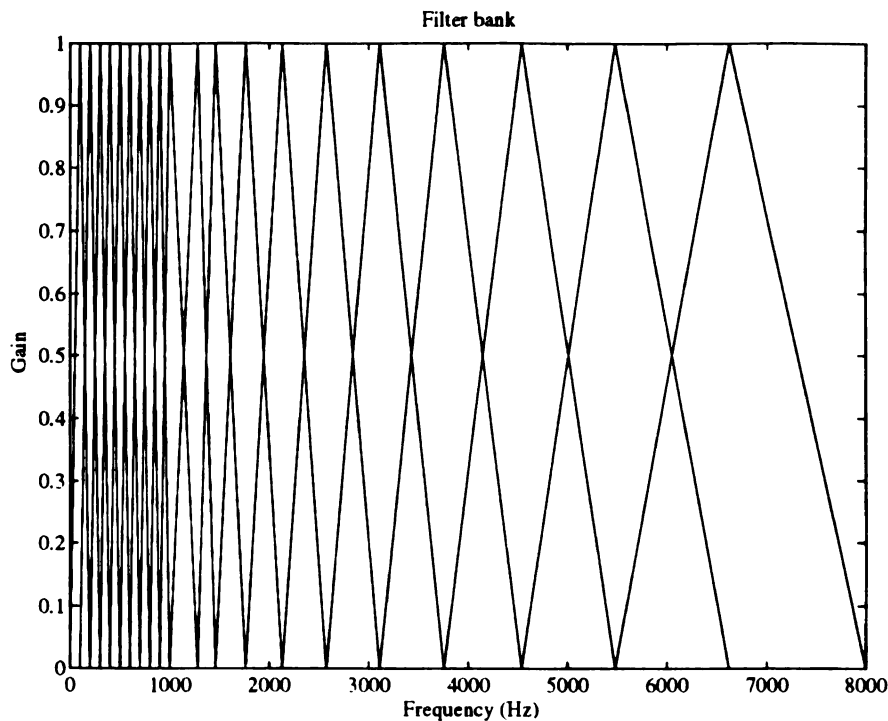


Figure 4.26: Filter bank for generating mel-based cepstral coefficients.

Codebook generation usually involves the analysis of a large sample of training sequences. Many iterative procedures have been proposed for designing codebooks [8], [67]. In this research, the *splitting* approach is adopted, which iteratively designs codebooks of increasing dimension using the *binary search* method. The binary search algorithm is summarized as follows: The set of training sequences in the  $K$ -dimensional space is first partitioned into two regions using the 2-means algorithm [67]. Then each of the two regions is divided into two subregions. This process is repeated until the space is divided into  $L$  regions. The *centroid* is computed for each region by averaging all of the training vectors in the partition. Then, the centroid of each region is a prototype vector in the VQ codebook. In our implementation,  $L$  is set to 128.

In order to improve the recognition rate, the power and differenced power are extracted from the speech signal. Power is computed from the speech waveform as follows:

$$P_i = \log\left(\sum_{k=1}^{N_w} f_k^2\right) \quad (4.11)$$

where  $P_i$  is the power for frame  $i$ . There are  $N_w$  discrete samples in each frame.  $N_w$  is set to 256 and consecutive frames are spaced 128 samples apart. The Hamming window is applied to the speech to create the frame samples.

It is also necessary to normalize for speaker loudness variation. In this work, we normalize by the factor which makes the variance of the power term unity. That is, each term is divided by the square root of its variance. Then, a direct distance measure can be computed without weighting the power term.

In addition to the power information, another source of information is differenced power, which is computed as follows:

$$D_i = P_{i+1} - P_{i-1} \quad (4.12)$$

where  $D_i$  is the differenced power term of the  $i^{th}$  frame of the utterance. Clearly, differenced power provides the information about relative changes in amplitude or loudness. Similarly,  $D_i$  is normalized to make the variance of the differenced power unity. The recognition accuracy improves significantly with the power and differenced power features added. Therefore, there are ten features selected for representing each frame of the speech signal including eight mel-based cepstral parameters, power, and

differenced power.

#### 4.2.3.3 HMM Training

Both the whole-word HMMs and phone models are used in the intraword level in the language graph, and methods described in this subsection are used to train either model form. The Bakis model [8] is used for both the word models and phone models. In the implementation, the three-state Bakis model is used to train the phone models, and six-state for the word models. The state transition coefficients are such that  $a_{ij} = 0, \forall j < i$  in a Bakis model. Further, additional constraints disallow jumps of more than two states. This constraint is of the form:  $a_{ij} = 0, \forall j > i + 2$ . For example, the form of the state transition matrix  $\mathbf{A}$  of Fig. 2.5 with six states is

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.13)$$

State 1 is chosen as the initial state so that the initial state probabilities are

$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases} \quad (4.14)$$

In this research, the *forward-backward* (or *Baum-Welch*) reestimation algorithm [8], [24], [21] is adopted for training both the word models and phone models. It has

been proved by Baum *et al.* [21] that either the parameters of the re-estimated model remain the same when a critical point is reached or every re-estimate is guaranteed to improve the model parameters. A detailed description of the forward-backward algorithm is given in Appendix E.

There are of total 1,156 words (with different phonetic labels) plus one silent “word” (representing the start vertex) and 60 phones trained in this experiment. These words are selected from the TIMIT database with at least five training data, and none of them is a function word. The training data for phone models are chosen from the “sx” sentences in the TIMIT database. The total number of training data for each phone model is at most 200. Figure 4.8 shows the actual number of phonetic segments in the training set.

Phone	Number	Phone	Number	Phone	Number
/iy/	2658	/r/	2953	/b/	1317
/ih/	2425	/w/	1216	/d/	1438
/eh/	1815	/y/	635	/g/	773
/ey/	1374	/hh/	462	/p/	1668
/ae/	1414	/hv/	337	/t/	2340
/aa/	1468	/el/	568	/k/	2463
/aw/	428	/s/	3679	/dx/	1081
/ay/	1198	/sh/	820	/q/	1626
/ah/	1282	/z/	2309	/jh/	787
/ao/	1162	/zh/	95	/ch/	597
/oy/	233	/f/	1308	/bcl/	1196
/ow/	944	/th/	476	/dcl/	2395
/uh/	263	/v/	1125	/gcl/	866
/uw/	342	/dh/	1387	/pcl/	1693
/ux/	956	/m/	2012	/tcl/	3454
/er/	1036	/n/	3565	/kcl/	2667
/ax/	2111	/ng/	736	/epi/	538
/ix/	4387	/em/	69	/pau/	363
/axr/	1490	/en/	402	/nx/	395
/ax-h/	225	/eng/	15	/l/	2615

Table 4.8: Phonetic segments in the training set.



### 4.2.3 Graph Partitioning

After the training process is completed, graph partitioning is used to select the key vertices in the language graph. A good partition set should satisfy the following criteria:

1. The vertices in the selected set  $\mathcal{C}$  are well-trained.
2. The selected set  $\mathcal{C}$  contains as few function words<sup>12</sup> as possible. These function words are problematic in continuous speech recognition [21] because they are articulated very poorly and are hard to recognize or even locate. For example, Table 4.10, Table 4.11, Table 4.12, and Table 4.13 enumerate the number of times the function words *the*, *in*, *with*, and *a*, respectively of different transcription labels in the TIMIT database.

To achieve a partition satisfying the criteria above, a non-negative *reward* for each vertex is required. A reward is assigned to each vertex according to the following policies:

1. Among all vertices representing whole words, higher rewards are assigned to vertices trained by more training data.
2. A higher reward is assigned to a vertex representing a whole-word HMM than to a vertex containing a phone model.

---

<sup>12</sup>Function words are typically prepositions, conjunctions, pronouns, and short verbs. For example, *the*, *a*, *in*, *with* are function words [21].

3. A penalty (negative reward) is assigned to a vertex representing a function word.

The function words used in this research are listed in Table 4.9. There are 42 function words indicated by Lee *et al.* [21].

a	all	and	any	are	at	be
been	by	did	find	for	from	get
give	has	have	how	in	is	it
list	many	more	of	on	one	or
show	than	that	the	their	to	use
was	were	what	why	will	with	would

Table 4.9: A list of function words appearing in this research.

Phonetic Labels	No.	Phonetic Labels	No.	Phonetic Labels	No.
dh-ix	688	dcl-d-iy	1	d-ih	2
dh-iy	298	q	3	th-ih	2
dh-ax	804	ah-z-ax	1	th-ax	2
dh-eh	25	dh-el	3	d-ah	1
dh-ih	86	s	3	z-ax	1
dh-ah	46	n-ax	1	z-ax-h	1
iy	66	n-iy	4	dh-q-eh	1
dh-ax-h	34	th-ax-h	5	dcl-dh-ix	1
dh	24	dh-aa	3	ax-pau-dh-ax	1
s-ax-h	1	dh-uh	3	sh-dh-iy	1
h-ix	5	z-dh-ix	1	th-ix	1

Table 4.10: Different ways the function word *the* is pronounced in TIMIT.

Clearly, a better vertex selection is a set of *higher* reward.

To quantify these requirements, a reward function of the following form is used for vertex  $x$ :

Phonetic Labels	No.	Phonetic Labels	No.	Phonetic Labels	No.
en	219	ih-ng	35	hv-ix-n	2
ih-n	261	ix-ng	100	ix-pau-q-ih-n	1
ax-n	114	er	61	q-ax-h-ng	1
q-ix-ng	40	q-ih-ng	11	hh-ix-ng	1
w-ix-n	23	iy-ng	1	ax-h-pau-q-ih-n	1
q-ih-n	86	ix-q-ih-n	2	hh-ih-n	1
eng	13	q-ih-eng	1	q-ih-m	1
ix-nx	128	ih-ax-n	2	hh-em	1
q-iy-ng	2	ih-ix-n	2	tcl	1
ax-hn	2	q-ih-nx	4		
ng	2	n	1		

Table 4.11: Different ways the function word *in* is pronounced in TIMIT.

Phonetic Labels	No.	Phonetic Labels	No.	Phonetic Labels	No.
w-ax-dh	19	w-ix-tcl-q	1	ix-th	2
w-ix-th	101	uh-dh	1	w-ix-s	8
w-ix-dh	25	w-ax-h-th	1	w-dh	1
w-ih-th	26	w-th	1	w-uh-tcl-t	1
w-ih-s	8	w-ih-ax	1	w-ix-tcl	2
w-ax-th	46	w-ih-tcl-th	1	w-ix-q	1
hh-w-ax-h-th	1	w-ih-dh	4		

Table 4.12: Different ways the function word *with* is pronounced in TIMIT.

Phonetic Labels	No.	Phonetic Labels	No.	Phonetic Labels	No.
ax-h	31	q-ax	42	q-eh	4
ax	497	q-ih	6	th	2
ix	401	eh	19	uh	6
ey	130	q-ix	12	hv-ey	1
q-ey	54	q-ax-h	1		
q-ah	35	ah	103		

Table 4.13: Different ways the function word *a* is pronounced in TIMIT.

$$\mathbf{Q}(x) = \alpha \mathbf{S}(x) + \beta \mathbf{T}(x) + \gamma(x) \mathbf{F}(x) \quad (4.15)$$

Let  $\mathbf{Q}(x)$  be the non-negative reward assigned to the vertex  $x$ . Let  $\mathbf{S}(x)$  be the weight for vertex  $x$  which is based on the number of occurrences of the corresponding word with its particular phonetic baseform in the TIMIT database. Generally speaking, the larger  $\mathbf{S}(x)$ , the more incoming and outgoing transitions from vertex  $x$ . Let  $\mathbf{T}(x)$  be the weight for the quality of training for the word at vertex  $x$ , e.g., the larger  $\mathbf{T}(x)$  is, the better training the resident word has. Let  $\mathbf{F}(x)$  be the penalty for the vertex  $x$  if it contains a function word.  $\alpha$ ,  $\beta$ , and  $\gamma(x)$  are the weighting factors. These factors satisfy the following properties:

$$\alpha + \beta = 1, \text{ and } \alpha > \beta \quad (4.16)$$

$$\gamma(x) = \begin{cases} -1, & \text{if the word at vertex } x \text{ is one of the function words in Table 4.5} \\ 0, & \text{otherwise} \end{cases} \quad (4.17)$$

Experimentally,  $\alpha$  is set to 0.6, and  $\beta$  is 0.4. The penalty for a function word  $\mathbf{F}(x)$  is assigned to be  $0.8\alpha\mathbf{S}(x)$ . The significance of  $\mathbf{S}(x)$  and  $\mathbf{T}(x)$  in the reward function is the following: Suppose the word model at vertex  $x$  is not well-trained. Then,  $\mathbf{T}(x)$  is set to zero, and  $\alpha\mathbf{S}(x)$  is the reward for the vertex  $x$ . If the word model at  $x$  is well-trained, then more reward can be assigned to  $x$  through  $\mathbf{T}(x)$ . At the same time,

the existence of  $\mathbf{T}(x)$  can be used to indicate the quality of training of the HMM for the word resident at  $x$ . By this, we mean that the more training data for training the model at  $x$ , the larger  $\mathbf{T}(x)$ . If a vertex is not represented by a well-trained word HMM, then it can be represented in terms of phone models.

After a reward is assigned to each vertex, the partitioning algorithm described in Chapter 2 is applied directly to the graph. Therefore, a set of vertices (i.e., those vertices in the set  $\mathbf{C}$ ) are selected for further evaluation and search. The partition results are affected by the following three factors:

1. The selection of the source vertex  $s$ . In a planar subgraph of the language graph, we need to choose a source vertex for breadth-first search (BFS) in Step 1 of the partitioning algorithm described in Section 2.3.1. Clearly, different source vertices for the BFS cause different partition results.
2. The reward assignment for each vertex. The rewards assigned to the vertices can be modified by applying different sets of weighting factors  $\alpha, \beta$ , and  $\gamma(x)$  in equation (4.17). The resulting partition depends on the choice of the weighting factors.
3. The extraction of planarity breaking arcs. For a nonplanar graph, the planarity breaking arcs are first extracted to create a planar subgraph. The graph partitioning algorithm for planar graphs is then applied to this planar subgraph to get a preliminary partition set. A complete partition set is achieved by applying a method described in Section 2.3.2. However, the resulting partition depends on which set of planarity breaking arcs is extracted. The set of arcs causing

nonplanarity is not unique.

Our goal is to find a partition which contains a high reward in the set  $\mathcal{C}$  by comparing several partitioning results. A set of high reward should be marked for evaluation. In our experiments, there are a total of 470 vertices ( $\approx 2.6\%$  of the total number of vertices in the language graph) in the set  $\mathcal{C}$ . Among the selected vertices, 191 of them are from the violation of the PST (recall the partitioning algorithm for nonplanar graphs in Chapter 2). The partition result is good enough because the set  $\mathcal{C}$  contains 25% of the vertex reward in the language graph, and there are a total of 217 well-trained word models in the selected set. In the next section, these selected vertices are used to test the performance of the graph search algorithm developed in this work.

#### 4.2.4 Two-Pass Graph Search

In this section, a few parameters for the word-spotting algorithm described in Section 3.2.1 are assigned. Experimentally, three thresholds,  $\Delta_L$ ,  $\Delta_l$ , and  $\Delta_u$ , are set as follows to yield good spotting results: For each selected word model  $\mathbf{M}^p$ ,  $\Delta_L$  is set to  $\bar{D}_p - \sigma_p$ ,  $\Delta_l$  is set to  $10^{-8}$ , and  $\Delta_u$  is set to  $10^{-2 \times \Delta_L}$ , where  $\bar{D}_p$  and  $\sigma_p$  are estimated from the training data. The significance of these parameters is discussed in Chapter 3. The stack size  $q$  is virtually unlimited. Therefore, there is no hard pruning.

## 4.3 Experiments

### 4.3.1 Methods and Measures of Performance

Partitioned graph search techniques are applied to recognition of continuous-speech taken from the TIMIT [21] database. Comparisons of the results of the following approaches are made:

1. partitioned graph search.
2. left-to-right search employing *randomly* selected vertices of the same number as used in the partitioned graph search cases.
3. conventional left-to-right stack search with pruning.

To compare the performance between the partitioned-search-based algorithms and the conventional left-to-right methods, the measures of performance include:

1. computational complexity.
2. recognition accuracy.
3. noise robustness.

In continuous-speech recognition, there are three types of errors: *substitution* errors (a word is misrecognized in the sentence), *deletion* errors (a correct word is omitted in the recognized sentence), and *insertion* errors (an extra word is added in the recognized sentence). Measures suggested by Lee [21] are used to assess the recognition performance. We first align the recognized word string against the correct

word string, and then compute the number of words correct, and the number of substitutions, deletions, and insertions. Five results are reported in computing the error rate. These are the *percent correct* (% Corr.), *percent substitutions* (% Subs.), *percent deletions* (% Dels.), *percent insertions* (% Ins.), and *word accuracy* (Word Acc.). Note that the percent correct does not consider insertions as errors but the word accuracy does. These performance measures are computed as follows:

$$\% \text{ Corr.} = 100 \times \frac{\text{Correct}}{\text{Correct Sent Length}} \quad (4.18)$$

$$\% \text{ Subs.} = 100 \times \frac{\text{Subs}}{\text{Correct Sent Length}} \quad (4.19)$$

$$\% \text{ Dels.} = 100 \times \frac{\text{Dels}}{\text{Correct Sent Length}} \quad (4.20)$$

$$\% \text{ Ins.} = 100 \times \frac{\text{Ins}}{\text{Correct Sent Length}} \quad (4.21)$$

$$\text{Error Rate} = 100 \times \frac{\text{Ins} + \text{Subs} + \text{Dels}}{\text{Correct Sent Length}} \quad (4.22)$$

$$\text{Word Acc.} = 1 - \text{Error Rate} \quad (4.23)$$

Since  $\text{Correct Sent Length} = \text{Correct} + \text{Subs} + \text{Dels}$ ,



$$\text{Word Acc.} = 100 \times \frac{\text{Correct} - \text{Ins}}{\text{Correct Sent Length}}. \quad (4.24)$$

To consider the measure of performance regarding to noise robustness, we need to know the *signal-to-noise ratio* (SNR). In this work, the SNR is computed as follows: Denote the SNR in dB by  $\kappa$ . Then,

$$\kappa(\text{dB}) = 10 \times \log \frac{\sum_i f_i^2}{\sum_i n_i^2} \quad (4.25)$$

where  $f_i$  is the sample at time  $i$  in the speech signal, and  $n_i$  the noise at time  $i$ . The summations in equation (4.27) are taken over the ranges of  $i$ 's which are corrupted by noise.

### 4.3.2 Results and Discussion

A set of 20 utterances taken from the dr3 dialect (North Midland) segment of TIMIT is used to evaluate the performance of the three methods listed above.

In order to reduce the computational effort for the left-to-right search, a pruning strategy is used to prune unlikely partial paths after evaluating at least the first and second models on each path. The partition evaluation likewise comprises a sort of “pruning” procedure and it is assured that no path is pruned in the partition search case unless at least two evaluations have taken place on it.

Noisy utterances in these experiments were created by adding noise of 0 dB SNR to randomly selected time intervals in the normal speech. The method used to corrupt

the normal speech is as follows: Four fixed length blocks (3,000 samples<sup>13</sup>) of noisy samples are added to the normal speech beginning at four randomly (using a C-language version of the machine-dependent random number generator) selected times.

The experimental results are shown in Table 4.14. In addition to the simple measure of performance defined above, we have added three further measures of performance specifically dealing with complexity reduction.

<b>Results Measures</b>	<b>Method</b>	<b>Partition</b>		<b>Random</b>		<b>Left-to-Right</b>	
		<b>Normal</b>	<b>Noisy</b>	<b>Normal</b>	<b>Noisy</b>	<b>Normal</b>	<b>Noisy</b>
<b>% Corr.</b>		79.05 %	56.08 %	59.46 %	40.54 %	60.14 %	42.56 %
<b>% Subs.</b>		16.22 %	35.13 %	34.46 %	52.03 %	39.19 %	50.68 %
<b>% Dels.</b>		4.73 %	8.79 %	6.08 %	7.43 %	0.67 %	6.67 %
<b>% Ins.</b>		0.00 %	2.03 %	2.70 %	4.05 %	6.76 %	6.67%
<b>Word Acc.</b>		79.05 %	54.05 %	56.76 %	36.49 %	53.38 %	35.81 %
<b>Average # of surviving paths after first PGS or LTR pruning</b>		763	778	1,327	1,318	976	991
<b># of vertices evaluated at first PGS or LTR pruning</b>		470	470	470	470	10,061	10,061
<b>Average node evaluated/ node pruned in the first and second time slots</b>		0.05271	0.05285	0.05824	0.05814	1.17034	1.173512

Table 4.14: Experimental results.

To assess complexity, let us consider the computational effort exerted to reduce the graph by pruning in the first pass of the search. As a reasonable *complexity*

<sup>13</sup>The average sentence length is about 45,000 samples.

*measure (CM)*, we define

$$\text{Complexity Measure (CM)} = \frac{\text{Average number of vertex evaluations performed}}{\text{Number of vertices pruned in the first two time slots}}. \quad (4.26)$$

In the equation above, the number of vertices pruned is obtained either effectively in the partition case, or directly in left-to-right search. This measure favors the left-to-right search in the sense that left-to-right evaluation works directly with the first two models on any path whereas the partitioned search can generally only affect them indirectly. The CM result for each noise-free utterance is shown in Fig. 4.27. Fig. 4.28 is a “blowup” of the CM comparison between the partition search and random selection methods so that the results can be seen clearly. Similar plots for the noisy speech are shown in Figs. 4.29 and 4.30. On the average, the computational expense of left-to-right search as measured by the CM is about 22 times more than the computational effort for partitioned search.

In considering this CM result, it should be carefully noted that the partition and search procedure is only performed *once* in these experiments. If partition and search were to be repeated in successive stages, even greater cost savings could be obtained. However, in a relatively small graph like the one under consideration here, the benefit of repartitioning is not great, especially in terms of the CM. For example, we estimate that the graph is reduced to about 10% of its original size by the initial partitioning procedure (requiring 470 evaluations). If a second partition recursion could reduce the remaining subgraph by an additional 90% using, say, 47 further evaluations, then the CM measure for partitioned search would be reduced only in the third decimal

place. The recursive partitioning procedure will become increasingly more important and beneficial as the graph size increases.

It is acknowledged that the recognition rates for each method explored in Table 4.14 are disappointingly low compared with rates obtained for other speaker-independent continuous-speech recognition systems using the TIMIT and other databases for evaluation [21], [65]. The main focus of this research has been to show the potential for a steep decrease in the computational effort using partitioned graph search compared with conventional left-to-right search approaches in the continuous-speech recognition task. Much of the research effort has focused on the graph-theoretic aspects of partitioning, and future attention to modeling and training issues, as well as more extensive experimental data, is expected to bring the recognition results in line with the state of the art. There is nothing about the partitioned search method which precludes similar modeling and training procedures to those used in very successful systems (see, e.g. [8]).

As an example of the effects of possibly unfavorable modeling and relatively few test utterances in the present work, consider the following facts. Three of the test utterances were found to be particularly problematic in the recognition task. These are sx140 (from fkms0), si908 (from fpkt0) and si1490 (from fkms0). We suspect that poorly trained models were to blame for prematurely pruning correct paths in each trial and in each method. Table 4.15 shows the results for the basic measures of performance with these sentences removed from the study. Clearly these results bring the present study more in line with expected rates. Much is still to be learned about important modeling issues, parameter settings, and other details through future

experimental research.

<b>Results</b> <b>Measures</b>	<b>Method</b>	<b>Partition</b>		<b>Random</b>		<b>Left-to-Right</b>	
		<b>Normal</b>	<b>Noisy</b>	<b>Normal</b>	<b>Noisy</b>	<b>Normal</b>	<b>Noisy</b>
<b>% Corr.</b>		92.8 %	65.6 %	69.6 %	47.2 %	70.4 %	50.4 %
<b>% Subs.</b>		5.6 %	26.4 %	26.4 %	46.4 %	29.6 %	42.4 %
<b>% Dels.</b>		1.6 %	8.0 %	4.0 %	6.4 %	0.0 %	7.2 %
<b>% Ins.</b>		0.0 %	0.8 %	0.0 %	4.0 %	4.8 %	7.2 %
<b>Word Acc.</b>		92.8 %	64.8 %	69.6 %	43.2 %	65.6 %	43.2 %
<b>Average # of surviving paths after first PGS or LTR pruning</b>		710	748	1,244	1,234	1,013	1,045
<b># of vertices evaluated at first PGS or LTR pruning</b>		470	470	470	470	10,061	10,061
<b>Average node evaluated/ node pruned in the first and second time slots</b>		0.05224	0.05258	0.05725	0.05736	1.17788	1.18487

Table 4.15: Experimental results with three problematic test utterances removed.

Finally, we wish to exhibit an overall measure of performance for the various methods which takes into account the complexity benefits of the partitioned search. Using CM as defined in (4.26) as a measure of computational effort, then the word accuracy normalized to CM is used as a good indicator of overall performance of these search methods:

$$\text{Normalized Word Accuracy (NWA)} = \frac{\% \text{ Word Accuracy}}{\text{Complexity Measure}}. \quad (4.27)$$

The results of the NWA for each search method are shown in Fig. 4.31. For the normal speech, the recognition results by partitioned graph search are significantly better

than the results of conventional left-to-right methods in these experiments. Furthermore, the partitioned graph search is more robust than the left-to-right approach in noisy environments. An interesting and expected outcome is that the robustness to noise afforded by the partitioned search is apparently due to the “scattered site” evaluation which takes place. This is evident in Fig. 4.31 because the random selection technique is also much more robust to noise than left-to-right search. However, previous results clearly indicate that random selection is not nearly as effective in reducing the computational complexity as partitioning. This is due to the fact that partitioning provides a very systematic way of selecting “scattered,” but high-payoff, vertices.

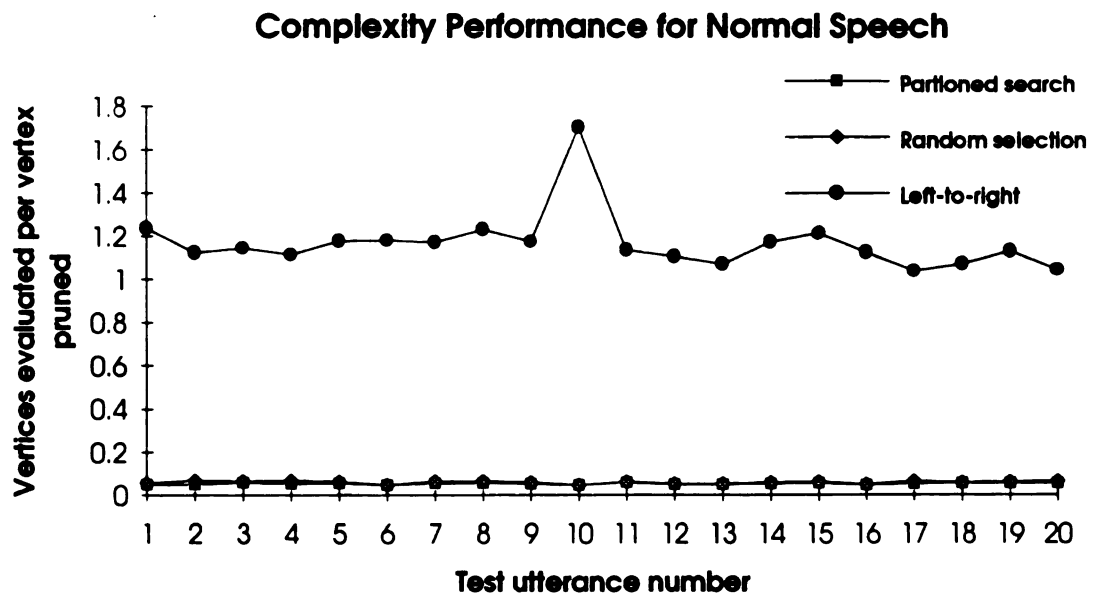


Figure 4.27: The complexity measure (CM) for normal speech using three different methods.

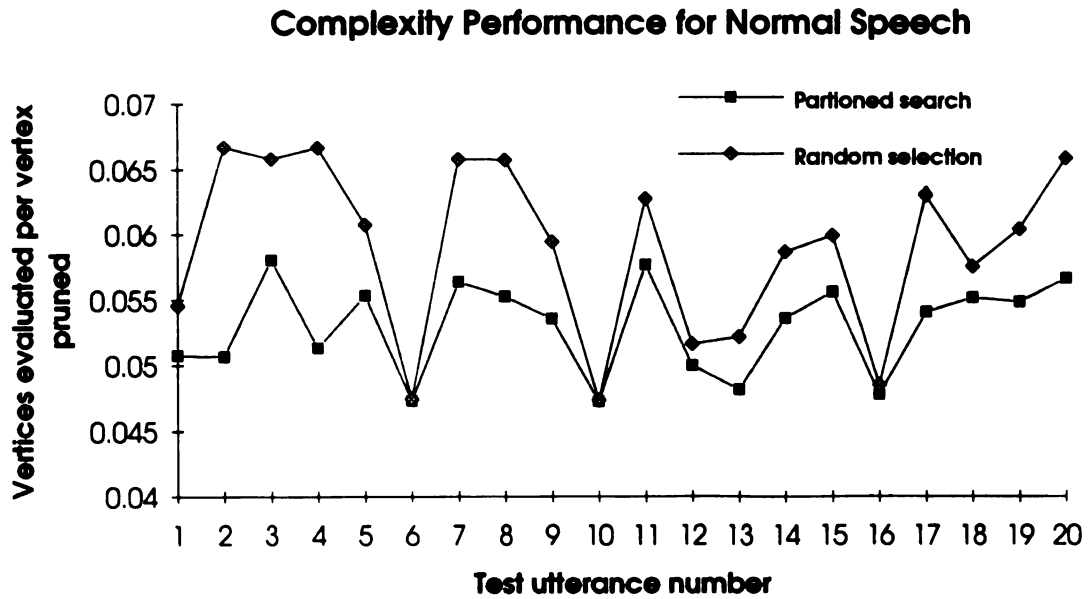


Figure 4.28: The complexity measure (CM) for normal speech using using partition and random selection methods.

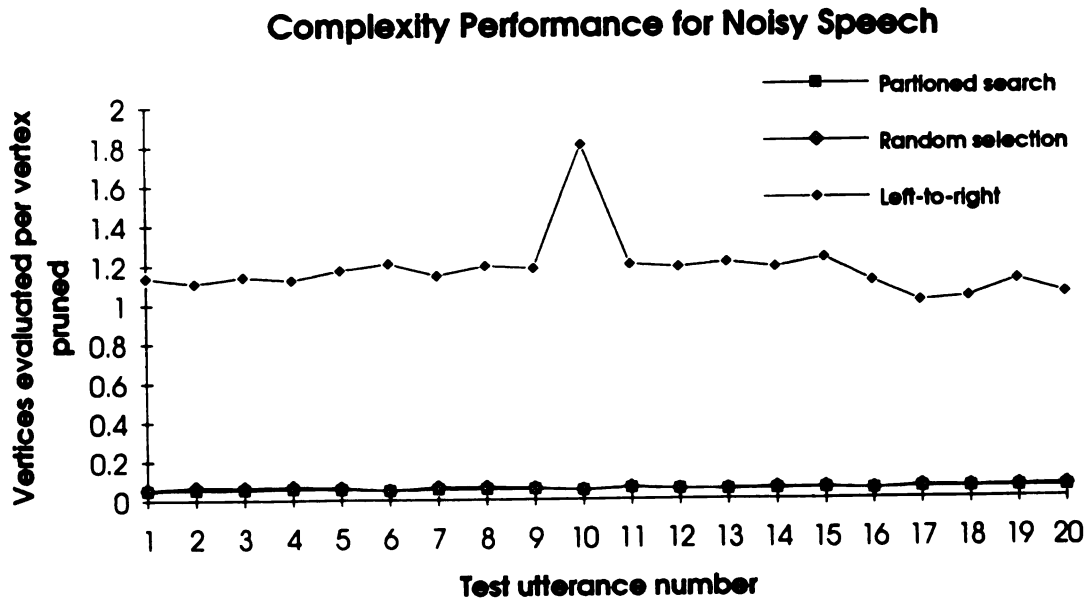


Figure 4.29: The complexity measure (CM) for noisy speech using three different methods.

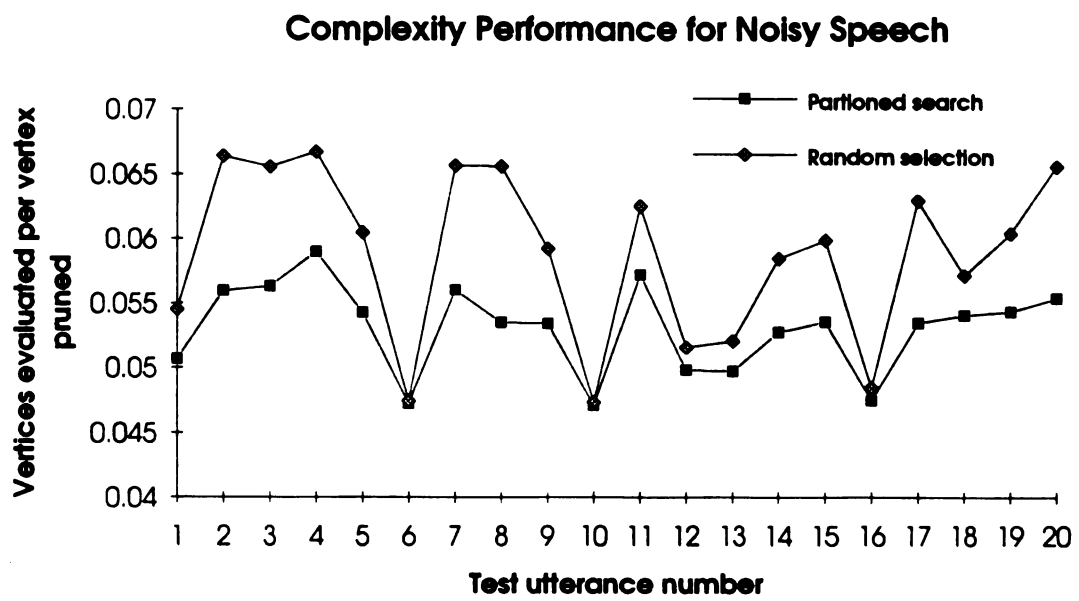


Figure 4.30: The complexity measure (CM) for noisy speech using using partition and random selection methods.



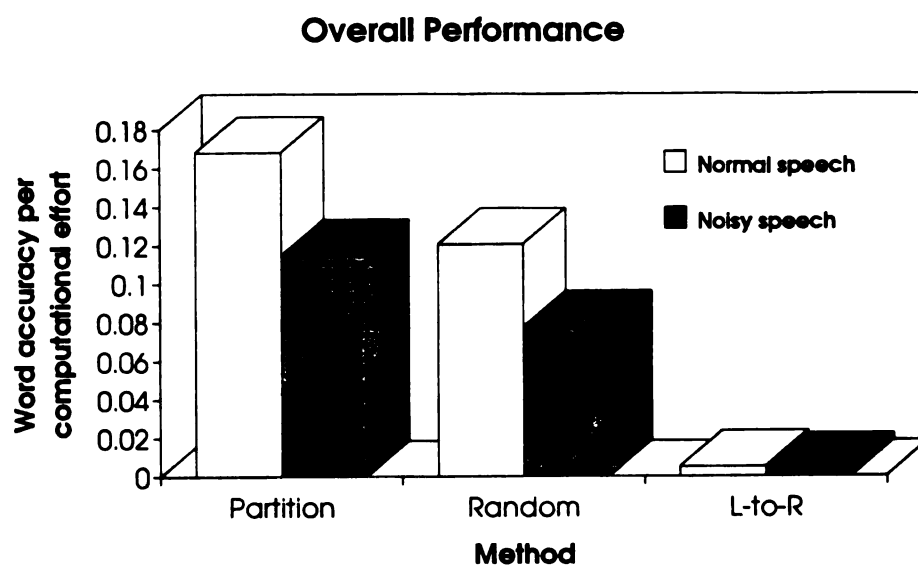


Figure 4.31: The overall performance as measured by the normalized word accuracy (NWA) of three different search methods.

# Chapter 5

## Further Discussion, Conclusions and Future Work

---

### 5.1 Summary and Further Discussion

This work has introduced a novel approach for continuous-speech recognition based on partitioned search of language graphs. In general, the most important finding of this work is that the partitioned search technique shows great potential for application to a large-vocabulary continuous-speech recognition. The general goal of this work has been to open the door to new research directions for future systems, which, if they are to be capable of recognizing speech based on practical and flexible languages, must involve very large graphs of very high perplexity. The central motivation for this partitioning operation is to reduce search complexity from  $\mathcal{O}(N)$  to  $\mathcal{O}(\sqrt{N})$  by a judicious selection of vertices. This complexity reduction has the obvious advantage of permitting the search of larger spaces, but also some less obvious benefits which are discussed above and below. To summarize, the advantages of the partitioned graph search procedures explored in this work are as follows:

- The decoding procedure is based upon a very large graph (reflecting a large language model). The benefits of a large graph are twofold:
  1. A large graph will reflect the original grammar more accurately thereby improving recognition performance.
  2. Models in a large graph will be trained in context and reflect the acoustic knowledge more accurately.
- These two “large model” benefits do not come at the expense of insufficient training because the partitioning procedure selects well-trained vertices. This feature permits a more accurate language model and better training in context.
- The computational complexity of partitioned search (with respect to conventional procedures) is greatly reduced by the inherent divide-and-conquer procedure.
- Partitioned search can be used to increase robustness to frequently misarticulated words such as function words.
- Partitioned search is more robust to impulsive or intermittent noise than conventional left-to-right search, because the evaluation of the data, by focusing on the partitioned  $C$ -set of vertices, is distributed across the observations in time. This means that the search is less likely to be pruned in the presence of unmodeled noise.

Significantly, partitioned search may be generally viewed as a type of  $n$ -best method, which has become quite popular in the speech recognition community in

recent years. An  $n$ -best approach employs a computationally inexpensive pass to pare down the problem to one involving a smaller graph of likely solutions. The smaller graph can then be subjected to intense scrutiny to find an optimal solution.

## 5.2 Contributions

The partitioning concept is simple to understand, but finding a good partition is a highly nontrivial problem. Graph partitioning algorithms developed in this work offer an efficient and systematic method for locating the high-payoff set of vertices for distributed evaluation. The implementation of the novel partitioning algorithms are reported in this dissertation.

In continuous-speech recognition tasks, the boundaries in the observation data are generally unknown. Without knowledge of the boundaries for each word in an utterance, the recognition task becomes significantly more challenging. In this work, a two-pass search algorithm for the unknown boundaries case is developed.

The major contributions of this research are summarized as follows:

1. A vertex partitioning algorithm for generally nonplanar graphs is developed, and the key operation for finding a cycle for the planar graph subpartitioning is improved.
2. Partitioned graph search algorithms, which are applicable to the case of unknown boundaries in the observation string, are developed.
3. Partitioned graph search techniques are applied to recognition of continuous-

speech taken from the TIMIT [21] database. Comparisons of the results using the following approaches for recognizing continuous speech are made:

- (a) partitioned graph search.
- (b) left-to-right search employing *randomly* selected vertices of the same number as used in the partitioned graph search cases.
- (c) conventional left-to-right stack search with pruning.

### 5.3 Future Work

The following issues for future research have emerged from this work:

1. There is a “bottleneck” in the process of partitioning a very large graph. The difficult task appears in the planarity testing operation. Before a language graph is partitioned, certain planarity breaking arcs need to be extracted, and the planar embedding for a large graph is extremely time and space consuming. To solve this problem when a large vocabulary system is considered, the following method is suggested. The language graph is decomposed into several small subgraphs and each subgraph is partitioned individually. Then the individual partitioning results are combined to form a selected set for the large graph. However, the selected vertex set  $C$  may no longer satisfy  $\mathcal{O}(\sqrt{N})$ . Therefore, the very interesting and challenging problem of finding more efficient and effective methods remains for future work.

2. Since most language models are expected to be representable only by nonplanar graphs, it is critical that the partitioning procedure be applicable to such graphs. One *ad hoc* method for dealing with the nonplanarity issue is described in Chapter 2. Other methods for treating nonplanarity and general effects of nonplanarity upon performance remain important issues for future research.
3. In the present research, resident at each vertex is either a whole-word HMM or a phone-based model composed of context-independent phone HMMs. Whole-word models are often found to outperform phone-based models if sufficient data are available to train them [81]. However, this training approach is impractical in a large speech recognition task. In the future systems, other models for representing these vertices are required, such as context-dependent phone models (e.g., triphones [21]).
4. Much is still to be learned about important modeling issues, parameter settings, and other details through future experimental research aimed at improving the overall performance of the partitioned graph search technique.
5. The complexity reduction is very significant when  $N$  is much larger than  $\sqrt{N}$ . Future systems must employ a very large ( $10^9 - 10^{12}$  vertices or even more) graphs of very high perplexity if they are to be capable of recognizing speech based on practical and flexible languages. In order to prevent the intractable problem of training of all vertices in huge graphs, the partitioning process can be used to determine a small set of high-payoff vertices to be trained. In this case, the partitioning is not guided by the availability of training data, but a

number of partition sets can be tried to find a set for which sufficient training data are available.

# Appendices



# Appendix A

## Graph-Theoretic Notations and Definitions

---

A graph  $G(\mathbf{V}, \mathbf{E})$  is an ordered pair consisting of a finite set of vertices  $\mathbf{V}$  and a finite-set of edges  $\mathbf{E}$ . Let  $|\mathbf{V}|$  denote the number of vertices in  $G$  and  $|\mathbf{E}|$  the number of edges. If each edge is an unordered pair of distinct vertices, then the graph is *undirected*. If each edge has an assigned orientation, then the graph is *directed*. Let  $(u, v)$  be an edge in a directed graph. Then  $(u, v)$  is said to join vertex  $u$  to vertex  $v$ ;  $u$  is the *tail* of  $(u, v)$ , and  $v$  is its *head*. If  $G$  is a directed graph, the *underlying graph* of  $G$  is the undirected graph formed by converting each edge of  $G$  to an undirected edge. Only undirected graphs are considered in the following definitions. Of course, we may extend the definitions to directed graphs by considering their underlying graphs.

A *walk* from  $v_0$  to  $v_k$  in  $G$  is a finite nonempty sequence whose terms are alternately vertices  $v_i$  and edges  $e_{i-1,i}$ ,  $1 \leq i \leq k$ , such that  $e_{i-1,i} = (v_{i-1}, v_i)$ . If all the vertices of a walk are distinct, then the walk is called a *path*. On the other hand, if all the edges of a walk are distinct, the walk is called a *trail*. A walk from a vertex to itself is called a closed walk. A *cycle* is defined as a closed trail whose origin and internal vertices are distinct.

A vertex  $w$  is *reachable* from a vertex  $v$  if there is a path from  $v$  to  $w$ . A graph  $G(V, E)$  is said to be *connected* if any vertex in  $G$  is reachable from any other vertex. If  $G$  is a connected graph, then a *spanning tree* of  $G$  is a subgraph  $T \subseteq G$  which is a tree and contains all the vertices of  $G$ . The *radius* of a tree is the maximum distance of any vertex from the root. A graph  $G(V, E)$  is said to be *planar* if it can be drawn, or embedded in the plane in such a way that the edges of the embedding intersect only at their ends. A planar embedding of a planar graph is sometimes referred to as a *plane graph* [1]. A planar representation of a graph divides the plane into a number of connected regions called *faces*, each bounded by some edges of the graph. The *boundary* of a face is a closed walk. A face  $f$  is said to be *incident* with the vertices and edges on its boundary. Figure A.32 depicts the faces of a particular embedding of the graph. Let  $b(f)$  denote the boundary of the face  $f$ . For example,  $f_2$  of the plane graph given in Figure A.32 is:

$$b(f_2) = v_1 e_{1,2} v_2 e_{2,7} v_7 e_{7,6} v_6 e_{6,1} v_1$$

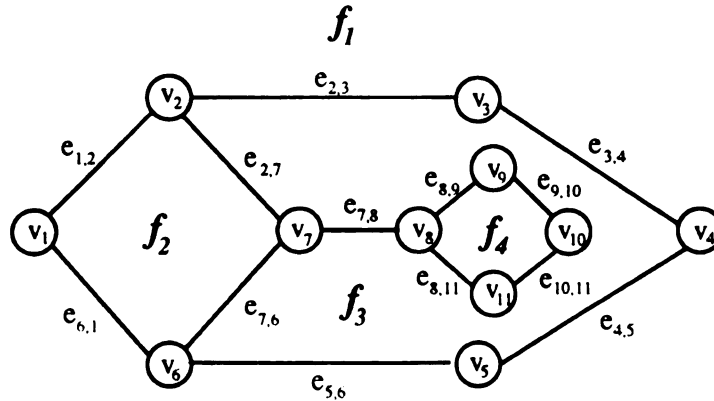


Figure A.32: A plane graph with four faces.

Of course, any planar representation of a connected planar graph always contains one face enclosing the graph. This face, called the *exterior face*, is  $f_1$  in Figure A.32. Note that if  $e$  is a *cut edge* in a plane graph, then only one face is incident with  $e$  since the removal of the cut edge disconnects the plane graph; otherwise, there are two faces incident with  $e$ . Similarly, a *cut vertex* of a connected graph is defined as a vertex whose removal disconnects the graph. For instance,  $e_{7,8}$  is a cut edge and  $v_7$  a cut vertex in Figure A.32. Since  $e_{7,8}$  is a cut edge, only  $f_3$  is incident with it. On the contrary,  $e_{1,2}$  is not a cut edge: there are exactly two faces ( $f_1$  and  $f_2$ ) incident with it. This property is important in the developments of graph partitioning algorithms described in Chapter 2. Finally, if  $G$  is connected and contains no cut vertices, then  $G$  is *biconnected*.

---

# Appendix B

## Supporting Lemmas for the PST

---

The constructive proof of the Planar Separator Theorem (PST) depends on two fundamental lemmas. A brief description of these supporting lemmas is as follows:

**Lemma B.1** *Let  $G$  be any  $N$ -vertex connected planar graph having non-negative vertex rewards summing to no more than one. Suppose  $G$  has a spanning tree of radius  $r$ . Then the vertices of  $G$  can be partitioned into three sets  $A$ ,  $B$ , and  $C$ , such that no edge joins a vertex in  $A$  with a vertex in  $B$ , neither  $A$  nor  $B$  has total reward exceeding  $2/3$ , and  $C$  contains no more than  $2r + 1$  vertices, one the root of the tree.*

The proof of the lemma proceeds by first embedding  $G$  in the plane and finding a breadth-first spanning tree [4] [28] of  $G$ . Since each face is triangulated by adding some additional edges, any nontree edge (including the new added edges) forms a simple cycle with some of the tree edges. Therefore, the length of this cycle is at most  $2r + 1$  if it contains the root of the tree. By the Jordan Curve Theorem [1], the cycle divides the graph into two parts, the inside and the outside of the cycle. Lipton and Tarjan [17] show by examples that at least one such cycle separates the graph so that neither the inside nor the outside of the cycle has total reward exceeding  $2/3$ .

**Lemma B.2** *Let  $G$  be any  $N$ -vertex connected planar graph. Suppose the vertices of  $G$  are partitioned into levels according to their distance from some vertex  $s$  and that  $L(l)$  denotes the total number of vertices on level  $l$ . Given any two levels  $l'$  and  $l''$  such that the total reward on levels 0 through  $l' - 1$  not exceeding  $2/3$  and levels  $l'' + 1$  and above have total reward not exceeding  $2/3$ , it is possible to find a partition  $A, B, C$  of the vertices of  $G$  such that no edge joins a vertex in  $A$  with a vertex in  $B$ , neither  $A$  nor  $B$  has total reward exceeding  $2/3$ , and  $C \leq L(l') + L(l'') + \max\{0, 2(l' - l'' - 1)\}$ .*

The lemma is very important for constructing a vertex partitioning algorithm for actual implementation. The proof of the lemma concerns the relationship between levels  $l'$  and  $l''$ . (i) Suppose  $l' \geq l''$ . Then the lemma is obviously true if we choose all the vertices on level  $l'$  to be in the set  $C$  and let  $A$  be all the vertices below the level  $l'$  and  $B$  be all the vertices above the level  $l'$ . (ii) Suppose that  $l' < l''$ . Since the vertices in levels  $l'$  and  $l''$  are deleted, the graph naturally partitions into three parts: vertices on levels 0 through  $l' - 1$ , vertices on  $l' + 1$  through  $l'' - 1$ , and vertices on levels  $l'' + 1$  and above. To find an appropriate vertex partitioning in this condition, two cases must be considered. One is the case in which the total reward between  $l' + 1$  and  $l'' - 1$  does not exceed  $2/3$ . A proper partition is obtained by setting  $A$  the part of the three with the most vertices,  $B$  the remaining two parts, and  $C$  the set of vertices in levels  $l'$  and  $l''$ . The other case is that in which the total reward between  $l' + 1$  and  $l'' - 1$  exceeds  $2/3$ . In this case, the part between  $l' + 1$  and  $l'' - 1$  requires subpartitioning. A subpartitioning is carried out as follows: All vertices on levels  $l''$  and above are deleted and all vertices on levels 0 through  $l' - 1$  shrunk to a

single vertex  $x$  of reward zero. A new graph, say  $\mathbf{G}'$ , is formed. Note that the new graph preserves planarity [17]. Apply Lemma B.1 to the new graph. Let  $\mathbf{A}'$ ,  $\mathbf{B}'$ ,  $\mathbf{C}'$  be its vertex partition; the set  $\mathbf{C}'$  being the vertices on the cycle. Therefore, a proper vertex partitioning of the graph  $\mathbf{G}$  derives from letting  $\mathbf{A}$  be the set among  $\mathbf{A}'$  and  $\mathbf{B}'$  with more vertices,  $\mathbf{C}$  the vertices in levels  $l'$  and  $l''$  plus the set  $\mathbf{C}'$ , and  $\mathbf{B}$  the remaining vertices.

# Appendix C

## Lipton-Tarjan Approach to Finding a Cycle

---

The Lipton-Tarjan (L&T) approach [17] for finding a cycle to complete the vertex partitioning for planar graphs involves the following steps:

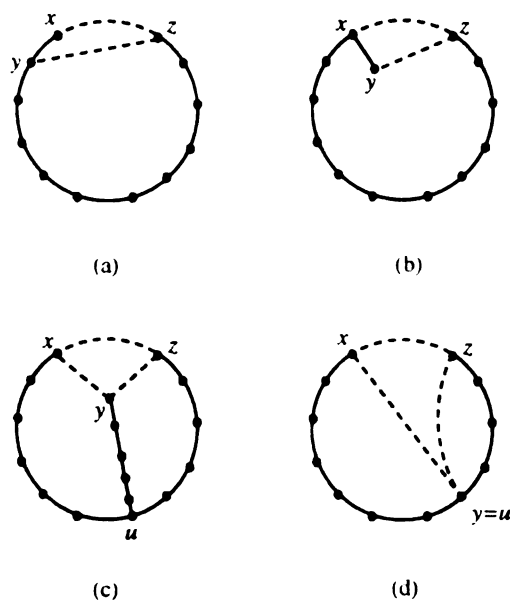


Figure C.33: Cases of Step 4 in the L&T approach for finding a proper cycle. Solid edges are tree edges; dotted edges are nontree edges.

**Step 1** Construct a breadth-first spanning tree rooted at  $x$  (created by shrinking procedure as described in the proof of Lemma B.2) in the new graph  $G'$ . Record, for

each vertex  $v$ , the parent of  $v$  in the tree, and the total number of all descendants of  $v$  including  $v$  itself.

**Step 2** Embed the new graph  $G'$  in the plane. Let  $H'$  be the planar embedding of  $G'$ . Make each face of  $H'$  a triangle by adding a suitable number of additional edges.

**Step 3** Choose any nontree edge, say  $(v_1, w_1)$ , and locate the corresponding cycle. Compute the number of vertices on each side of this cycle by scanning the tree edges incident on each side of the cycle. Determine which side of the cycle has greater reward and call it the “inside.”

**Step 4** Let  $(x, z)$  be the nontree edge whose cycle is the current candidate to complete the separator. If the reward inside the cycle exceeds  $2/3$ , then find a better cycle by the following method. Locate the triangle  $(x, y, z)$  which has  $(x, z)$  as a boundary edge and lies inside the  $(x, z)$  cycle. Two cases must be considered:

1. If either  $(x, y)$  or  $(y, z)$  is a tree edge, let  $(v, w)$  be the nontree edge among  $(x, y)$  and  $(y, z)$ . Then compute the reward inside the  $(v, w)$  cycle. This case is shown in Figure C.33 (a) and (b).
2. If neither  $(x, y)$  nor  $(y, z)$  is a tree edge, determine the tree path from  $y$  to the  $(x, z)$  cycle by following parent pointers from  $y$ . Let  $u$  be the vertex on the  $(x, z)$  cycle reached during this search. Then, compute the number of vertices inside the  $(x, y)$  cycle and  $(y, z)$  cycle. Let  $(v, w)$  be the nontree edge among  $(x, y)$  and  $(y, z)$  whose cycle has more reward inside it. This case is shown in Figure C.33 (c) and (d).



Repeat Step 4 until a cycle satisfying Lemma B.1 is found.

# Appendix D

## A Planarity Testing Algorithm

---

The planarity testing algorithm of Demoucron *et al.* [1] is shown in Figure D.34. Before using this algorithm to determine whether a given graph is planar, some pre-processing considerably simplifies the work. Note the following points:

1. If the graph is not connected, then each component should be subjected to planarity testing.
2. If no cycle is found, then the graph is a tree. Therefore, it is planar.
3. If  $|\mathbf{E}| < 9$  or  $N < 5$ , then the graph must be planar; if  $|\mathbf{E}| > 3N - 6$ , then the graph must be nonplanar [1].

The following definitions are required: Let  $\mathbf{G}_i(\mathbf{V}_i, \mathbf{E}_i)$  be a subgraph of  $\mathbf{G}$ , a *bridge*  $\mathbf{B}$  of  $\mathbf{G}$  related to  $\mathbf{G}_i$  is then:

1. *either* an edge  $(u, v) \in \mathbf{E}$  where  $(u, v) \notin \mathbf{E}_i$  and  $(u, v) \in \mathbf{V}_i$ , *or*
2. a connected component of  $(\mathbf{G} - \mathbf{G}_i)$  plus any edge incident with this component.

We denote by  $V(\mathbf{B}, \mathbf{G}_i)$  the *vertices of attachment* of  $\mathbf{B}$  to  $\mathbf{G}_i$ . Let  $\mathbf{H}_i$  be an embedding of  $\mathbf{G}_i$  in the plane. If  $\mathbf{B}$  is any bridge of  $\mathbf{G}_i$ , then  $\mathbf{B}$  is said to be *drawable* in a face  $f$  of  $\mathbf{H}_i$  if  $V(\mathbf{B}, \mathbf{G}_i)$  are contained in the boundary of  $f$ . We write  $\mathbf{F}(\mathbf{B}, \mathbf{H}_i)$  for the sets of faces of  $\mathbf{H}_i$  in which  $\mathbf{B}$  is drawable. The algorithm to follow is based on a very important criterion: *If  $\mathbf{F}(\mathbf{B}, \mathbf{H}_i) = \emptyset$ , then we cannot obtain further planar subgraph embedding. Thus the algorithm will terminate for nonplanarity.*

Given a graph  $\mathbf{G}$ , the algorithm determines an increasing sequence  $\mathbf{G}_1, \mathbf{G}_2, \dots$  of planar subgraphs of  $\mathbf{G}$ , and corresponding planar embeddings  $\mathbf{H}_1, \mathbf{H}_2, \dots$  when  $\mathbf{G}$  is planar. Through the algorithm, it is easy to record the faces of each subgraph  $\mathbf{G}_{i+1}$  at each iteration  $i$  as shown in Figure D.34. The procedure is as follows:

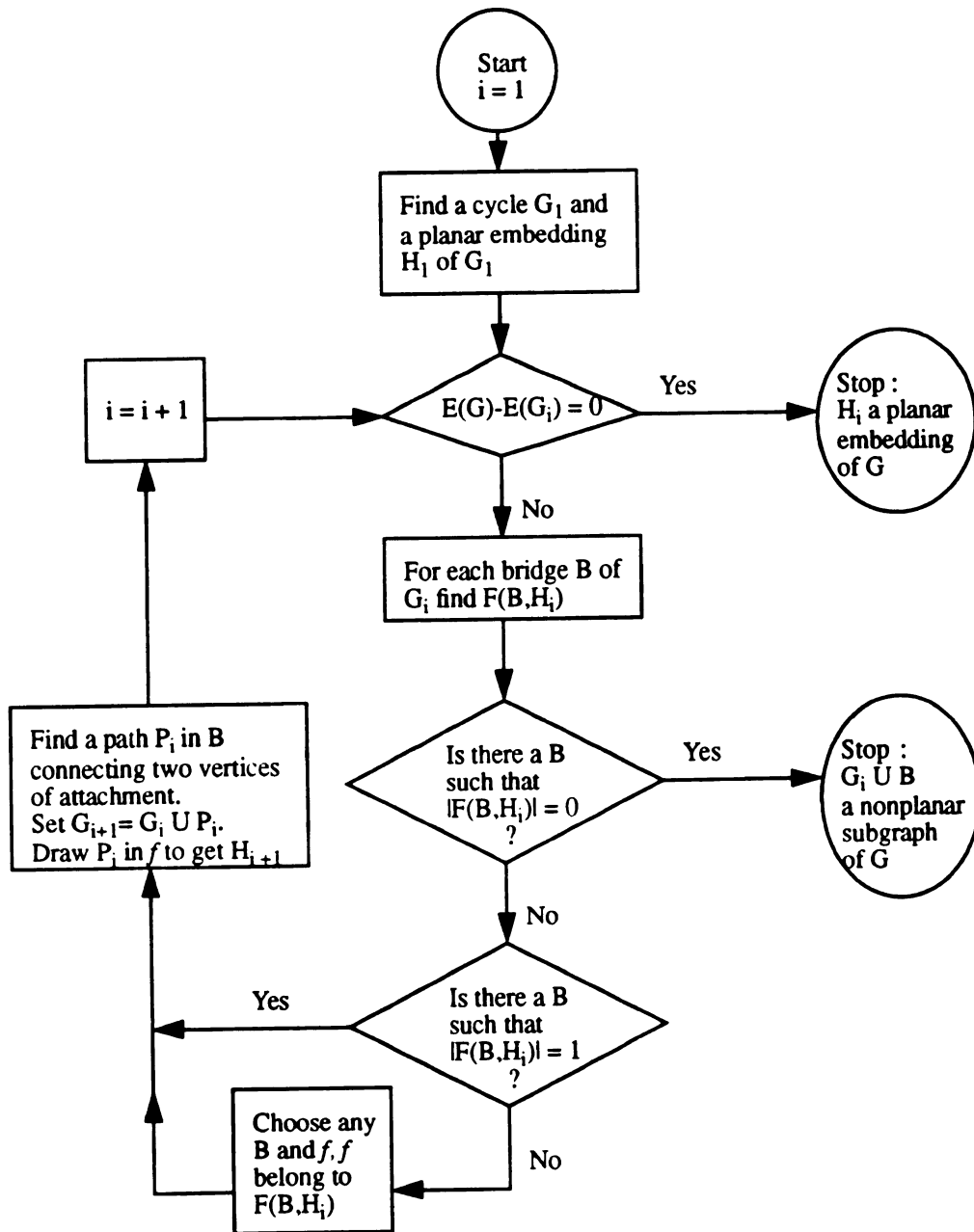
1. If there exists a bridge  $\mathbf{B}$  such that  $\mathbf{F}(\mathbf{B}, \mathbf{H}_i) = \emptyset$ , then the graph  $\mathbf{G}$  is non-planar. Thus, the planarity testing and planar embedding ceases.

2. If there exists a bridge  $\mathbf{B}$  such that  $|\mathbf{F}(\mathbf{B}, \mathbf{H}_i)| = 1$ , then let  $f = \mathbf{F}(\mathbf{B}, \mathbf{H}_i)$ .

From the bridge  $\mathbf{B}$ , choose a path  $P_i \subseteq \mathbf{B}$  and set  $\mathbf{G}_{i+1} = \mathbf{G}_i \cup P_i$ . Thus, the faces of  $\mathbf{G}_{i+1}$  can be obtained by drawing  $P_i$  in the face  $f$  of  $\mathbf{G}_i$ .

3. Otherwise, choose any face  $f$  and any bridge  $\mathbf{B}$  such that  $f \in \mathbf{F}(\mathbf{B}, \mathbf{H}_i)$ . From the bridge  $\mathbf{B}$ , choose a path  $P_i \subseteq \mathbf{B}$  and set  $\mathbf{G}_{i+1} = \mathbf{G}_i \cup P_i$ . The faces of  $\mathbf{G}_{i+1}$  can also be obtained by drawing  $P_i$  in the face  $f$  of  $\mathbf{G}_i$ .

Note that if  $\mathbf{G}$  is planar, then by *Euler's formula* [1],  $|\mathbf{E}| - N + 2$  faces will be found. Since these faces have been found following implementation of the process shown in Figure D.34, a fixed planar embedding of the graph  $\mathbf{G}$  can be constructed.

Figure D.34: The planarity testing algorithm of Demoucron *et al.*

# Appendix E

## Elements of the Hidden Markov Model

---

In speech recognition, hidden Markov models (HMM) have become increasingly popular in the last fifteen years since the modeling provides a mathematically rigorous approach and works very well in practice [8], [9], [21], [24]. Much of the popularity of the HMM is based on the discovery of efficient methods of estimating its parameters. The form of the HMM can be chosen according to the knowledge of application domain and the parameters trained from known data.

To use HMM, three basic problems (training, evaluation, and recognition) must be understood:

1. The training problem: Training in HMM is simply the estimation of the parameters of a model from a set of known training data (observation sequences) in which the parameters can best describe how a given observation sequence is generated.
2. The evaluation problem: Given a model and a sequence of observations, the evaluation problem considers the probability that the observed sequence is produced by the model.

3. The recognition problem: Recognition in HMM is the identification of an unknown observation sequence by choosing the most likely model (representing a word, or a sentence, etc.) that produces the observation sequence.

A discrete observation HMM, say  $\mathbf{M}$ , is defined by a set of  $I$  states,  $J$  observation symbols, and three probabilistic matrices:

$$\mathbf{M} = \{\Pi, \mathbf{A}, \mathbf{B}\}. \quad (\text{E.1})$$

Let the individual states be denoted as  $S = 1, 2, \dots, I$ , and the observation symbols  $Z = \{z_1, z_2, \dots, z_J\}$ . Then, the probabilistic matrix  $\Pi$  is the initial state probability:

$$\Pi = \{\pi_i\}, \quad \pi_i = P(\text{state } i \text{ at } t = 1), \quad (\text{E.2})$$

where the variable  $t$  denotes discrete time.  $\mathbf{A}$  is an  $I \times I$  matrix containing the Markovian state transition probabilities:

$$\mathbf{A} = \{a_{ij}\}, \quad a_{ij} = P(\text{state } j \text{ at } t + 1 | \text{state } i \text{ at } t), \quad (\text{E.3})$$

and  $\mathbf{B}$  is an  $I \times J$  matrix containing the observation symbol probability distributions:

$$\mathbf{B} = \{b_j(k)\}, \quad b_j(k) = P(z_k \text{ at } t | \text{state } j \text{ at } t). \quad (\text{E.4})$$

Note that  $a_{ij}$  and  $b_j(k)$  satisfy the following properties:

$$a_{ij} \geq 0, b_j(k) \geq 0, \quad \forall i, j, k \quad (\text{E.5})$$

$$\sum_j a_{ij} = 1, \forall i \quad (\text{E.6})$$

$$\sum_k b_j(k) = 1, \forall j. \quad (\text{E.7})$$

In the HMM, the state sequence is *hidden* but the output sequence can be directly observed. A set of sample model topologies is shown in Figure E.35. The HMM topology of Figure E.35(a) is called an *ergodic* topology because any state can be reached from any other state. However, this topology is inappropriate for speech recognition because speech consists of an ordered sequence of sounds. Figure E.35(b) through E.35(d) are all left-to-right models. The left-to-right restriction forces an ordering on the state sequence in which the path evaluation must either stay in the same state or go to a higher numbered state. Figure E.35(b) is the general left-to-right topology. Figure E.35(c), a *Bakis* model [9] (stay, move, or skip one), and Figure E.35(d), a *linear* (stay or move) model, are commonly used for speech recognition. Although we have dichotomized HMMs into ergodic and left-to-right models, there are still many possible variations and combinations [8], [9], [21].

Let  $\mathbf{Y} = y_1, y_2, \dots, y_T$  be the discrete observation of a training utterance for a model  $\mathbf{M}$ . Our goal is to choose the model parameters such that  $P(\mathbf{Y}|\mathbf{M})$  is locally maximized. Let  $\alpha_t(i)$  be the forward variable and  $\beta_t(i)$  be the backward variable defined as:

$$\alpha_t(i) = P(y_1, y_2, \dots, y_t, \text{ and } s_t = i | \mathbf{M}) \quad (\text{E.8})$$

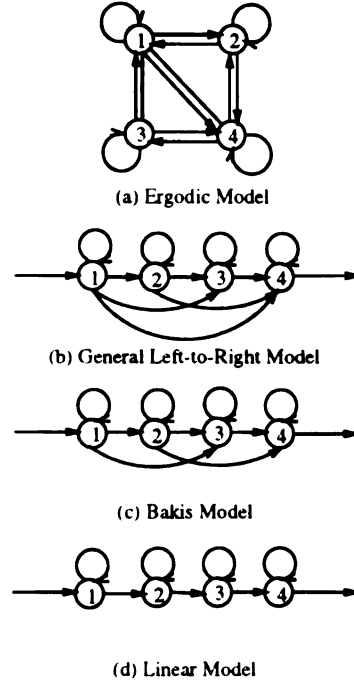


Figure E.35: Some examples of four-state HMM topologies.

$$\beta_t(i) = P(y_{t+1}, y_{t+2}, \dots, y_T | s_t = i, \mathbf{M}) \quad (\text{E.9})$$

where  $s_t$  denotes the state at time  $t$ . Define  $\varphi_t(i, j)$  the probability of being in state  $i$  at time  $t$ , and state  $j$  at time  $t + 1$  given the model  $\mathbf{M}$  and the observation sequence  $\mathbf{Y}$ . The forward-backward algorithm [24], which can be used to adjust the model parameters  $(\mathbf{A}, \mathbf{B}, \Pi)$  and maximize the probability of the observation sequence given the model, is sketched below:

**Step 1** *Initialize all the elements in  $\Pi$ ,  $\mathbf{A}$ , and  $\mathbf{B}$  with reasonable random numbers satisfying the basic properties discussed above.*

**Step 2** *For each training datum, implement Steps 3 to 5.*



**Step 3** Calculate the forward and backward variables inductively as follows:

$$\alpha_1(i) = \pi_i b_i(y_1), \quad 1 \leq i \leq I$$

$$\alpha_{t+1}(j) = \left[ \sum_{i=1}^I \alpha_t(i) a_{ij} \right] b_j(y_{t+1}), \quad 1 \leq t \leq T-1, \text{ and } 1 \leq j \leq I$$

$$\beta_T(i) = 1, \quad 1 \leq i \leq I$$

$$\beta_t(i) = \sum_{j=1}^I a_{ij} b_j(y_{t+1}) \beta_{t+1}(j) \quad t = T-1, T-2, \dots, 1, \text{ and } 1 \leq i \leq I.$$

**Step 4** Calculate  $\varphi_t(i, j)$  and  $\gamma_t(i)$ .

$$\varphi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(y_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^I \sum_{j=1}^I \alpha_t(i) a_{ij} b_j(y_{t+1}) \beta_{t+1}(j)}$$

$$\gamma_t(i) = \sum_{j=1}^I \varphi_t(i, j).$$

Then, a set of reestimation formulas for  $\Pi$ ,  $\mathbf{A}$ , and  $\mathbf{B}$  are:

$$\hat{\pi}_i = \gamma_1(i)$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \varphi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\hat{b}_j(k) = \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}.$$

**Step 5** Set  $\pi_i \leftarrow \hat{\pi}_i$ ,  $a_{ij} \leftarrow \hat{a}_{ij}$ , and  $b_j(k) \leftarrow \hat{b}_j(k)$ ,  $\forall i, j, k$ .

**Step 6** *If some convergence criterion is reached, then stop. Otherwise, go to Step 2.*

# Bibliography

# Bibliography

- [1] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, New York: American Elsevier Publishing, 1976.
- [2] F. Harary, *Graph Theory*, Reading, Massachusetts: Addison-Wesley, 1969.
- [3] S. Even, *Graph Algorithms*, Potomac, Maryland: Computer Science Press, 1979.
- [4] A. Gibbons, *Algorithmic Graph Theory*, New York: Cambridge University Press, 1985.
- [5] B.T. Lowerre and R. Reddy, "The HARPY speech understanding system," in W.A. Lea (ed.), *Trends in Speech Recognition*, pp.340–360, Englewood Cliffs, New Jersey: Prentice-Hall, 1980.
- [6] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Trans. on Information Theory*, vol. IT-13, pp. 260-269, Apr. 1967.
- [7] F. Jelinek, "Fast sequential decoding algorithm using a stack," *IBM J. of Research and Development*, vol. 13, pp. 675-685, Nov. 1969.
- [8] J.R. Deller, Jr., J.G. Proakis, and J.H.L. Hansen, *Discrete Time Processing of Speech Signals*, New York: Macmillan, 1993.
- [9] J. Picone, "Continuous speech recognition using hidden Markov models," *IEEE ASSP Magazine*, pp. 26–41, July 1990.
- [10] D.B. Paul, "Speech recognition using hidden Markov models," *Lincoln Laboratory J.*, vol. 3, no. 1, pp. 41–61, Spring 1990.
- [11] L.R. Bahl, F. Jelinek and R.L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 2, pp. 179–190, Mar. 1983.
- [12] J.R. Deller, Jr. and R.K. Snider, "Reducing redundant computation in HMM evaluation," *IEEE Trans. on Speech and Audio Processing*, June, 1993.

- [13] C.G. Venkatesh, J.R. Deller, Jr., and C.C. Chiu, "A graph partitioning approach to signal decoding," manuscript.
- [14] L.R. Bahl and F. Jelinek, "Decoding for channels with insertions, deletions and substitutions with applications to speech recognition," *IEEE Trans. on Information Theory*, vol. IT-21, no. 4, pp.404-411, July 1975.
- [15] S.E. Levinson, "Structural methods in automatic speech recognition," *Proc. of the IEEE*, vol. 73, no. 11, 1985.
- [16] C.C. Chiu, *Algorithms for Signal Decoding Using Graph Partitioning*, (M.S. thesis), Dept. of Electrical Engineering, Michigan State University, East Lansing, 1991.
- [17] R.J. Lipton and R.E. Tarjan, "A separator theorem for planar graphs," *SIAM J. of Computing*, vol. 36, no. 2, pp. 177-189, Apr. 1979.
- [18] H.N. Djidjev, "On the problem of partitioning planar graphs," *SIAM J. of Algebraic and Discrete Methods*, vol 3, no. 2, pp. 229-240, June 1982.
- [19] G.L. Miller, "Finding small simple cycle separator for 2-connected planar graphs," *Proc. 16th Annual ACM Symp. on Theory of Computing*, pp. 376-382, Apr. 1984.
- [20] C.C. Chiu, C.G. Venkatesh, A.-H. Esfahanian, and J.R. Deller, Jr., "An algorithm for planar graphs subpartitioning," submitted to *J. of Combinatorial Mathematics and Combinatorial Computing*.
- [21] K.F. Lee, *Automatic Speech Recognition, the Development of the SPHINX System*, Boston: Kluwer Academic Publishers, 1989.
- [22] D.A. Plaisted, "A heuristic algorithm for small separators in arbitrary graphs," *SIAM J. of Discrete Mathematics*, vol 19, no. 2, pp. 267-280, Apr. 1990.
- [23] S. Rao, "Finding near optimal separators in planar graphs," *Proc. 28th Annual IEEE Symp. on Foundations of Computer Science*, pp. 225-237, 1987.
- [24] L.R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. of the IEEE*, vol. 77, no. 2, 1989.
- [25] C.H. Lee and L.R. Rabiner, "A frame-synchronous network search algorithm for connected word recognition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 11, pp. 1649-1658, 1989.
- [26] R. Schwartz and Y.L. Chow, "The  $N$ -best algorithm: an efficient and exact procedure for finding the  $N$  most likely sentence hypotheses," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 81-84, 1990.

- [27] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 28, pp. 357–366, 1980.
- [28] J.A. McHugh, *Algorithmic Graph Theory*. Englewood Cliffs, New Jersey: Prentice-Hall, 1990.
- [29] "Getting started with the DARPA TIMIT CD-ROM: An acoustic-phonetic continuous speech database," National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, 1988.
- [30] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, Massachusetts: Addison-Wesley, 1974.
- [31] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling, Volume I: Parsing*, Englewood Cliffs, New Jersey: Prentice-Hall, 1972.
- [32] E.B. Messinger, L.A. Rowe, and R.R. Henry, "A divide-and-conquer algorithm for the automatic layout of large directed graphs," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 21, no. 1, pp. 1–12, Jan./Feb. 1991.
- [33] M.R. Fellows, "Transversals of vertex partitions in graphs," *SIAM J. of Discrete Mathematics*, vol 3, no. 2, pp. 206–215, May 1990.
- [34] E.R. Barnes, A. Vannelli, and J.Q. Walker, "A new heuristic for partitioning the nodes of a graph," *SIAM J. of Discrete Mathematics*, vol 1, no. 3, pp. 299–305, Aug. 1988.
- [35] T. Ozawa, "The principal partition of vertex-weighted graphs and its applications," *Discrete Algorithms and Complexity*, pp. 5–33, 1987.
- [36] J.P. Hutchinson and G.L. Miller, "On deleting vertices to make a graph of positive genus planar," *Discrete Algorithms and Complexity*, pp. 81–98, 1987.
- [37] H. Gazit and G.L. Miller, "A parallel algorithm for finding a separator in planar graphs," *Proc. 28th Annual IEEE Symp. on Foundations of Computer Science*, pp. 238–248, 1987.
- [38] J.R. Gilbert, D.J. Rose, and A. Edenbrandt, "A separator theorem for chordal graphs," *SIAM J. of Algebraic and Discrete Methods*, vol 5, no. 3, pp. 306–313, Sep. 1984.
- [39] R.E. Tarjan, "Space-efficient implementations of graph search methods," *ACM Trans. on Mathematical Software*, vol 9, no. 3, pp. 326–339, Sep. 1983.
- [40] E.R. Barnes, "An algorithm for partitioning the nodes of a graph," *SIAM J. of Algebraic and Discrete Methods*, vol 3, no. 4, pp. 541–550, Dec. 1982.

- [41] N. Deo, G.M. Prabhu, and M.S. Krishnamoorthy, "Algorithms for generating fundamental cycles in a graph," *ACM Trans. on Mathematical Software*, vol 8, no. 1, pp. 26–42, Mar. 1982.
- [42] H.N. Djidjev, "A linear algorithm for partitioning graphs," *Comptes rendus de l'Academie Bulgare des Sciences*, vol. 35, pp. 1053–1056, 1982.
- [43] R.J. Lipton and R.E. Tarjan, "Applications of a planar separator theorem," *SIAM J. of Computing*, vol. 9, no. 3, pp. 615–627, Aug. 1980.
- [44] J. Hopcroft and R.E. Tarjan, "Efficient planarity testing," *J. of the Association for Computing Machinery*, vol. 21, no. 4, pp. 549–568, Oct. 1974.
- [45] J.R. Gilbert, J.P. Hutchinson and R.E. Tarjan, "A separator theorem for graphs of bounded genus," *J. of Algorithms*, vol. 5, pp. 391–407, 1984.
- [46] R.C. Read, "A new method for drawing a planar graph given the cyclic order of the edges at each vertex," Research Report CORR 86-14, University of Waterloo, July 1986.
- [47] R.J. Lipton and R.E. Tarjan, "Applications of a planar separator theorem," *Proc. 18th Annual IEEE Symp. on Foundations of Computer Science*, pp. 162–170, 1977.
- [48] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms*, Cambridge, Massachusetts: MIT Press, 1991
- [49] G. Brassard and P. Bratley, *Algorithms – Theory and Practice*, Englewood Cliffs, New Jersey: Prentice-Hall, 1988.
- [50] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, Massachusetts: Addison-Wesley, 1974.
- [51] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, New York: W.H. Freeman, 1979.
- [52] R. Sedgewick, *Algorithms*, Reading, Massachusetts: Addison-Wesley, 1988.
- [53] F.S. Roberts, *Applied Combinatorics*, Englewood Cliffs, New Jersey: Prentice-Hall, 1984.
- [54] G. Casella and R.L. Berger, *Statistical Inference*, Belmont, California: Wadsworth, 1990.
- [55] P.E. Pfeiffer, *Concepts of Probability Theory*, New York: Dover Publications, 1978.
- [56] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill, 1984.

- [57] R. Schwartz and Y.L. Chow, "The N-best algorithm: an efficient and exact procedure for finding the N most likely sentence hypotheses," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 81–84, 1990.
- [58] D.B. Paul, "Algorithms for an optimal  $A^*$  search and linearizing the search in the stack decoder," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 693–696, 1991.
- [59] L.Bahl, P.S. Gopalakrishnan, D. Kanevsky and D. Nahamoo, "Matrix fast match: a fast method for identifying a short list of candidate words for decoding," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 345–348, 1989.
- [60] L.R. Rabiner and S.E. Levinson, "A speaker-independent, syntax-directed, connected word recognition system based on hidden Markov models and level building," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-33, no. 3, pp. 561–573, 1985.
- [61] H.Ney, "The use of one-stage dynamic programming algorithm for connected word recognition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-32, no. 2, pp. 263–271, 1984.
- [62] F. Jelinek, L. Bahl, and R.L. Mercer, "Design of a linguistic statistical decoder for the recognition of continuous speech," *IEEE Trans. on Information Theory*, vol. IT-21, no. 3, pp. 250–256, 1975.
- [63] C.C. Tappert, N.R. Dixon, and A.S. Rabinowitz, "Application of sequential decoding for converting phonetic to graph representation in automatic recognition of continuous speech (ARCS)," *IEEE Trans. on Audio and Electroacoustics*, vol. AU-21, pp. 225–228, 1973.
- [64] L.R. Bahl, P.de Souza, P.S. Gopalakrishnan, D. Kanevsky and D. Nahamoo, "Constructing groups of acoustically confusable words," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 85–88, 1990.
- [65] K.F. Lee and H.W. Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 37, no. 11, pp. 1641–1648, 1989.
- [66] P.S. Cohen and R.L. Mercer, "The phonological component of an automatic speech-recognition system," *IEEE Symp. on Speech Recognition*, pp. 177–187, 1974.
- [67] J. Makhoul, S. Roucos and H. Gish, "Vector quantization in speech coding," *Proc. of the IEEE*, vol. 73, no. 11, pp. 1551–1587, 1985.
- [68] R.M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. on Information Theory*, pp. 64–74, 1963.



- [69] L.R. Rabiner and R.W. Schafer, *Digital Processing of Speech Signals*, Englewood Cliffs, New Jersey: Prentice-Hall, 1978.
  - [70] J.D. Markel and A.H. Gray, Jr., *Linear Prediction of Speech*, Berlin, Heidelberg, New Jersey: Springer-Verlag, 1976.
  - [71] E.J. Yannakoudakis and P.J. Hutton *Speech Synthesis and Recognition Systems*, England: Ellis Horwood Limited, 1987.
  - [72] J.S. Bridle, M.D. Brown and R.M. Chamberlain, "An algorithm for connected word recognition," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 899–902, 1982.
  - [73] H. Sakoe, "Two-level DP-matching - A dynamic programming-based pattern matching algorithm for connected word recognition," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 27, no. 6, pp. 588–595, 1979.
  - [74] A.N. Ince, *Digital Speech Processing: Speech Coding, Synthesis and Recognition*, Boston: Kluwer Academic Publishers, 1992.
  - [75] S. Furui, *Digital Speech Processing, Synthesis, and Recognition*, New York: Marcel Dekker, Inc., 1989.
  - [76] D. O'Shaughnessy, *Speech Communication, Human and Machine*, New York: Addison-Wesley, 1988.
  - [77] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Communications*, vol. COM-28, no. 1, pp. 84–95, Jan. 1980.
  - [78] J.G. Wilpon, L.R. Rabiner, C.-H. Lee, and E.R. Goldman "Automatic recognition of keywords in unconstrained speech using hidden Markov models," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 38, no. 11, pp. 1870–1878, 1990.
  - [79] J.G. Wilpon, C.-H. Lee, L.R. Rabiner, and E.R. Goldman "Application of hidden Markov models for recognition of a limited set of words in unconstrained speech," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 254–257, 1989.
  - [80] J.G. Wilpon, L.G. Miller, and P. Modi "Improvements and applications for key word recognition using hidden Markov modeling techniques," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 309–312, 1990.
  - [81] L.R. Bahl, R. Bakis, P.S. Cohen, A.G. Cole, F. Jelinek, B.L. Lewis, and R.L. Mercer, "Recognition results with several experimental acoustic processors," *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 249–251, 1979.
-