THE

MICHIGAN STATE UNIVERSITY LIBRARIES

3 1293 00902 4757

This is to certify that the

thesis entitled

Algorithms For Signal Decoding Using Graph Partitioning.

presented by

Chuang-Chien Chiu

has been accepted towards fulfillment of the requirements for

Masters degree in Science

John R. Deller Jr.

Major professor

Date 21 Feb. 91

O-7639

MSU is an Affirmative Action/Equal Opportunity Institution

LIBRARY
Michigan State
University

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
SEP 26 1997		
MAR 5 1995		
06 1995		

MSU Is An Affirmative Action/Equal Opportunity Institution
c:\circ\datedue.pm3-p.1

ALGORITHMS FOR SIGNAL DECODING USING GRAPH PARTITIONING

By

Chuang-Chien Chiu

A THESIS

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

MASTER OF SCIENCE

Department of Electrical Engineering

1991

ABSTRACT

ALGORITHMS FOR SIGNAL DECODING USING GRAPH PARTITIONING

By

Chuang-Chien Chiu

This work is concerned with graph partitioning algorithm, which can be used to select $\mathcal{O}(\sqrt{N})$ nodes for evaluation from an N node decoding graph. The theoretical basis for this work is found in the paper by Venkatesh, Deller and Cozzens [1], and it is the primary purpose of this study to develop algorithms for implementing and testing the methods. The small number of nodes selected by the partition will cover a significant number of paths in the decoding graph, offering a cost-effective method for simultaneously evaluating multiple paths. When a pruning strategy is combined with partitioning, the overall graph search complexity is $\mathcal{O}(\sqrt{N})$. This represents a significant decrease with respect to conventional left-to-right decoding approaches which are usually $\mathcal{O}(N)$.

Theoretical and implementation issues involved in the development of computer algorithms for the partitioning procedure are the principal focus of this work. The algorithms are tested on a large graph (10^3 nodes, 1,200 edges) using a preliminary version of a "multiple stack" search procedure described in [1].

Copyrighted by

CHUANG-CHIEN CHIU

1991

ACKNOWLEDGEMENTS

The original idea for using graph partitioning techniques in signal decoding contained in this thesis was suggested by Professor John R. Deller, Jr., who is my thesis advisor, and Dr. C.G. Venkatesh in CTA Incorporated. I would like to express my gratitude to both of them for their invaluable guidance and helpful suggestions throughout this research. I am also gratefully indebted to Professor Abdol-Hossein Esfahanian and Professor Majid Nayeri, for their time and effort in discussing and reviewing my thesis.

A special word of thanks for my thesis advisor, Professor John R. Deller, Jr., for all his patience and encouragement and support over last year we have worked together. It is a very nice and unforgettable experience to be his student.

This work is supported in part by a grant from CTA Incorporated in Bedford, Massachusetts.

Contents

List of Tables

List of Figures

1 Introduction and Background

1.1 The Graph Search Problem in Signal Decoding

Graph theory has long been recognized as one of the most useful mathematical ways to model many real world problems. For signal decoding problems (e.g., speech recognition or image reconstruction), Venkatesh, Deller and Cozzens [1] have presented a graph-theoretic strategy for reducing the computational complexity with respect to conventional decoding approaches [2] [3]. The technique, which is based on the Planar Separator Theorem (PST) of Lipton and Tarjan [4], uses a partitioning approach to locate $\mathcal{O}(\sqrt{N})$ nodes for evaluation from an N node decoding graph G. Through the evaluation of this relatively small number of nodes, an optimal path for a given observation string is found. Venkatesh, *et al.* have worked out the theoretical details underlying this method, but no computer algorithms were developed for selecting these "high payoff" nodes of G. Therefore, the main purpose of this research has been to develop algorithms for partitioning and search of graphs in signal decoding.

Before presenting the graph partitioning and search algorithms to solve the signal decoding problem, it is important to sketch the underlying theory and discuss its advantages with respect to conventional left-to-right search. The following fact [5] [6] is fundamental to the methods:

Every planar graph with N vertices has a set of vertices $\mathcal{C}$ of size $\mathcal{O}(\sqrt{N})$ which separates the set of vertices $\mathcal{A}$ from the set of vertices $\mathcal{B}$, where $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ is a partition *of the vertices in the given planar graph and the size of $\mathcal{A}$ and $\mathcal{B}$ are no more than $\frac{2N}{3}$. The set $\mathcal{C}$ is called an $\mathcal{O}(\sqrt{N})$ separator.*

Since the nodes in $\mathcal{C}$ separate the graph into two sets of nodes $\mathcal{A}$ and $\mathcal{B}$, making it impossible to pass from one to the other without encountering $\mathcal{C}$, the nodes in $\mathcal{C}$ must contain many convergent and divergent paths. This condition is shown in Fig. 1. Thus, the nodes of $\mathcal{C}$ can be considered as "bottlenecks" in the graph into which many paths converge. The PST, therefore, guarantees the selection of significant nodes (those nodes in $\mathcal{C}$) which will cover many paths.

A pruning process [1] is generally included in a search procedure to minimize the number of overall node evaluations. This procedure introduces the risk of pruning the correct (most likely) path. The increased coverage provided by the partitioning procedure will generally provide an acceptable "pruning safety". The coverage issue is at the heart of the pruning safety factor disscussed above. On the other hand, $\mathcal{C}$ is relatively small ($\mathcal{O}(\sqrt{N})$), and these nodes are selected at distributed locations throughout G, rather than always "from the left" which is conventional [2] [3]. Thus, the use of this set can minimize the number of node evaluations. If the procedure for evaluating nodes is very costly, the partitioning approach will greatly improve the computational cost compared to a conventional left-to-right search.

Many computational problems on graphs can be performed more efficiently on planar graphs. We shall focus on the problem of finding an appropriate separator $\mathcal{C}$ of size $\mathcal{O}(\sqrt{N})$ in a *planar* decoding graph. The details for extending this method to nonplanar graphs are founded in [1] and developing appropriate algorithms for this more general case will be the subject of future research. After the node selection process is completed, the decoding graph will be searched and pruned according to a likelihood measure. A preliminary version of a multiple stack decoding algorithm developed by Deller and reported in [1] will be presented to carry out this procedure.

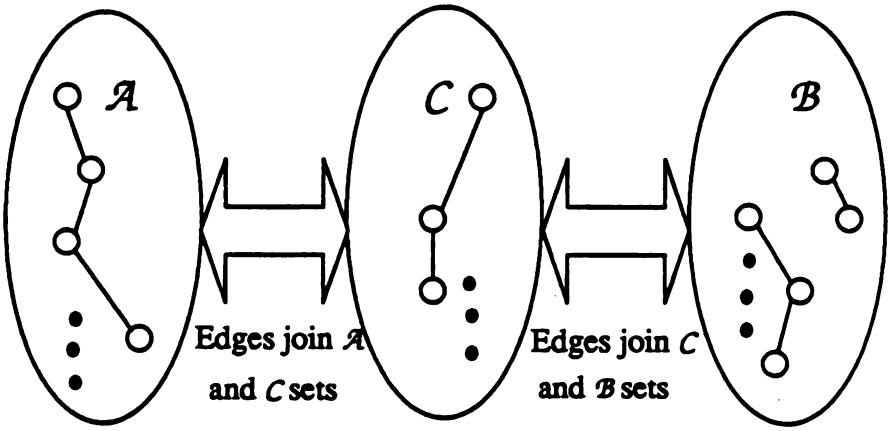


Figure 1: The condition implied by the Planar Separator Theorem. $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ form a vertex partition in a planar graph such that no edge joins a vertex in $\mathcal{A}$ with a vertex in $\mathcal{B}$.

1.2 The Outline of the Vertex Partitioning Algorithm

In 1979 Lipton and Tarjan [4] presented a sequential algorithm which takes $\mathcal{O}(N)$ time for finding a separator of size $\sqrt{8N}$ for planar graphs. The *Planar Separator Theorem* (PST) of Lipton and Tarjan is as follows:

Theorem 1 (Planar Separator Theorem [4]) *Let* $\mathbf{G}$ *be any N-vertex planar graph having non-negative vertex costs summing to no more than one. Then the vertices of* $\mathbf{G}$ *can be partitioned into three sets* $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, *such that no edge joins a vertex in* $\mathcal{A}$ *with a vertex in* $\mathcal{B}$, *neither* $\mathcal{A}$ *nor* $\mathcal{B}$ *has total cost exceeding* $\frac{2}{3}$, *and* $\mathcal{C}$ *contains no more than* $2\sqrt{2N}$ *vertices.*

The separator theorem described above is a general form pertaining to planar graphs which have nonnegative costs on the vertices. However, for the purpose of partitioning in signal decoding problems, the desired separator theorem is the special case of equal-cost vertices [4, Cor. 2]. The relevant corollary is as follows:

Corollary 1 *In any N-vertex planar graph* $\mathbf{G}$, *a subset of vertices* $\mathcal{C}$ *in* $\mathbf{G}$ *is a separator if remaining vertices can be partitioned into two sets* $\mathcal{A}$ *and* $\mathcal{B}$ *such that there are no edges from* $\mathcal{A}$ *to* $\mathcal{B}$, *and* $|\mathcal{A}|$, $|\mathcal{B}| \leq \frac{2N}{3}$. *Then the sets* $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ *form a partition of* $\mathbf{V}$, *and the separator* $\mathcal{C}$ *is of size* $\sqrt{8N}$. $|\mathcal{A}|$ *denotes the number of vertices in* $\mathcal{A}$ *and* $\mathbf{V}$ *is the set of all vertices.*

In the following discussion, all vertices are assigned equal cost values which sum to unity.

The algorithm for the theorem above (see [4]) requires a breadth-first search (BFS)[1] of the graph as input. Theoretically, given a BFS of a planar graph, a simple cycle separator can be found. This fact was shown in the proof of Lemma 2 in

[1] A BFS of a graph with respect to some vertex s is a labeling of the vertices such that the label of a vertex v is the shortest distance from s to v.

[4]. Even though their separator in general is not a simple cycle, according to the PST we still need to find a *simple* cycle to complete a satisfactory vertex partition in one special case (we will describe this special case in Chapter 3). Finding a simple cycle separator in a planar graph is an interesting and challenging task. In previous research, Miller [5] presented an algorithm to find a small cycle separator for a 2-connected[2] graph. However, decoding graphs generally will not be 2-connected. For actual implementation of the partitioning algorithm, finding an appropriate simple cycle to complete the vertex partitioning is essential.

In [4], Lipton and Tarjan suggest a planarity algorithm [7] to construct a planar embedding of the planar graph. Through the planar embedding, a vertex partition can be found which satisfies the Planar Separator Theorem. However, the description of the planarity algorithm [7] does not provide sufficient information for actual implementation. First, the planarity algorithm does not provide direct information for the determination of each face[3] of the planar graph, but the boundary of the faces is used to find a satisfactory simple cycle. Secondly, when a simple cycle has been formed, the total number of vertices on each side (inside and outside) of this cycle must be computed, and the determination that neither the inside nor the outside of this cycle has a total number of vertices exceeding $\frac{2N}{3}$ must be made. However, this task is not described in enough detail in their algorithm for actual implementation. In this work we have solved these problems and recommend an algorithm for actual implementation. The algorithm is shown in Fig. 2.

The method of Lipton-Tarjan for finding a planar separator was improved in 1982 by Djidjev [8], who obtained a separator of size $\sqrt{6N}$. Our algorithm is based

[2] A 2-connected graph is a connected graph which contains no cut vertices.

[3] A face in a planar embedding is a connected region bounded by edges and vertices; the boundary of a face is regarded as a closed walk.

on the solution of Djidjev with a smaller constant[4] to find a vertex partitioning. We make use of the planarity testing algorithm of Demoucron *et al.* [9] to find a planarity embedding of the planar graph. The boundary of each face is easily found by Demoucron's algorithm, and these faces are recorded for future reference. It should be noted that these faces can actually construct a planar representation of the graph G (see Section 2.2).

The general outline of the partitioning algorithm is presented here, and its correctness is described in detail later. Let G(V,E) be a planar graph. G consists of a set of vertices, $\mathbf{V} = \{v_i\}$, a set of edges, $\mathbf{E} = \{e_{kj}\}$ (e_{kj} connects vertex v_k to vertex v_j). N is the total number of vertices. We implement a BFS to partition the vertices into levels according to their distance from some vertex v. Let $L(l)$ be the total number of vertices on level l. For each $\alpha \in (0,1)$, let l_α denote a level such that $\sum_{l=0}^{l_\alpha-1} L(l) < \alpha N$ and $\sum_{l=0}^{l_\alpha} L(l) \geq \alpha N$. The outline of the algorithm is as follows:

Step 1 *Classify the vertices of* G *into levels according to their distance from some vertex v in* G.

Step 2 *Find two levels $l_{1/3}$ and $l_{2/3}$.*

Step 3 *If there exists a level l in the interval $[l_{1/3},\ l_{2/3}]$ such that $L(l) \leq \sqrt{6N}$, then the nodes in the C set are the vertices on level l. Stop. Otherwise, go to step 4.*

Step 4 *Find a nonnegative integer, say j, such that*

$$\sum_{l=l_{1/3}-j+1}^{l_{2/3}+j-1} L(l) < \frac{2N}{3},$$

[4] Lipton and Tarjan deduced the constant for the separator C is $\sqrt{8}$, Djidjev improved the constant to $\sqrt{6}$.

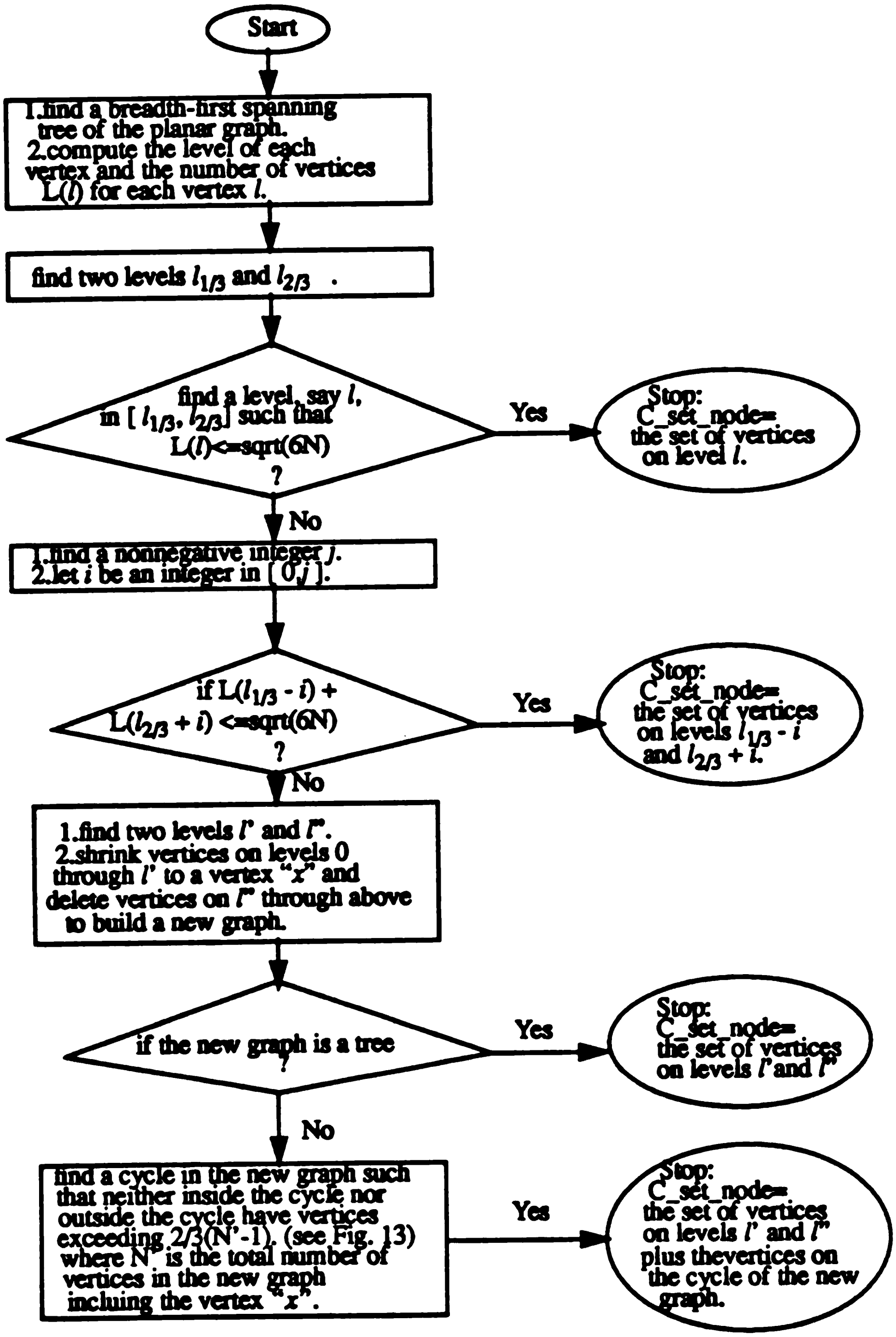


Figure 2: A partitioning algorithm for implementation.

$$\sum_{l=l_{1/3}-j}^{l_{2/3}+j} L(l) \geq \frac{2N}{3}$$

If there exists $i \in [0,j]$ *and* $L(l_{1/3}-i)+L(l_{2/3}+i) \leq \sqrt{6N}$*, then the nodes in the C set are the vertices on levels* $L(l_{1/3}-i)$ *and* $L(l_{2/3}+i)$*. Stop. Otherwise, go to step 5.*

Step 5 *Find two levels* l' *and* l'' *such that* l' *is the highest level and* l'' *is the lowest level which satisfies the following conditions:*

$$l' \leq l_{1/3}-j-1 < l_{2/3}+j+1 \leq l''$$

$$L(l')+2(l_{1/3}-j-1-l') \leq 2\sqrt{M}$$

$$L(l'')+2(l''-(l_{2/3}+j+1)) \leq 2\sqrt{N-P}$$

where $M=\sum_{l=0}^{l_{1/3}-j-1} L(l)$, $P=\sum_{l=0}^{l_{2/3}+j} L(l)$.

Step 6 *Shrink all the vertices on levels 0 through* l' *to a single vertex* x *and delete all the vertices on levels* l'' *and above to form a new graph, say* $\mathbf{G}'$.

Step 7 *Find a simple cycle separator, say C, of the new graph* $\mathbf{G}'$ *(The procedure will be presented in Chapter 3). The nodes in C set are the vertices on levels* l' *and* l'' *plus the vertices on the cycle C.*

In the following chapters, theoretic aspects of the partitioning graph search algorithms, and complete discussion of the algorithms for finding an optimal path in the decoding problem will be presented. Precise computer algorithms for the decoding problem will also be presented. In Chapter 2, the planarity testing algorithm is studied and a topological embedding of a graph in the plane for finding an appropriate vertex partition is found. A method for creating a planar decoding graph for experiments is given. In Chapter 3, $\sqrt{N}$-separator theorems are introduced and an algorithm for finding a vertex partitioning is provided. In Chapter 4, the graph search problem is discussed and a multiple stack decoding algorithm will be presented to carry out this partitioning graph search. Also, a large planar decoding graph is created for testing the graph partitioning and search method using a preliminary version of the multiple stack search procedure described in [1]. The flow of topic coverage is shown in Fig. 3.

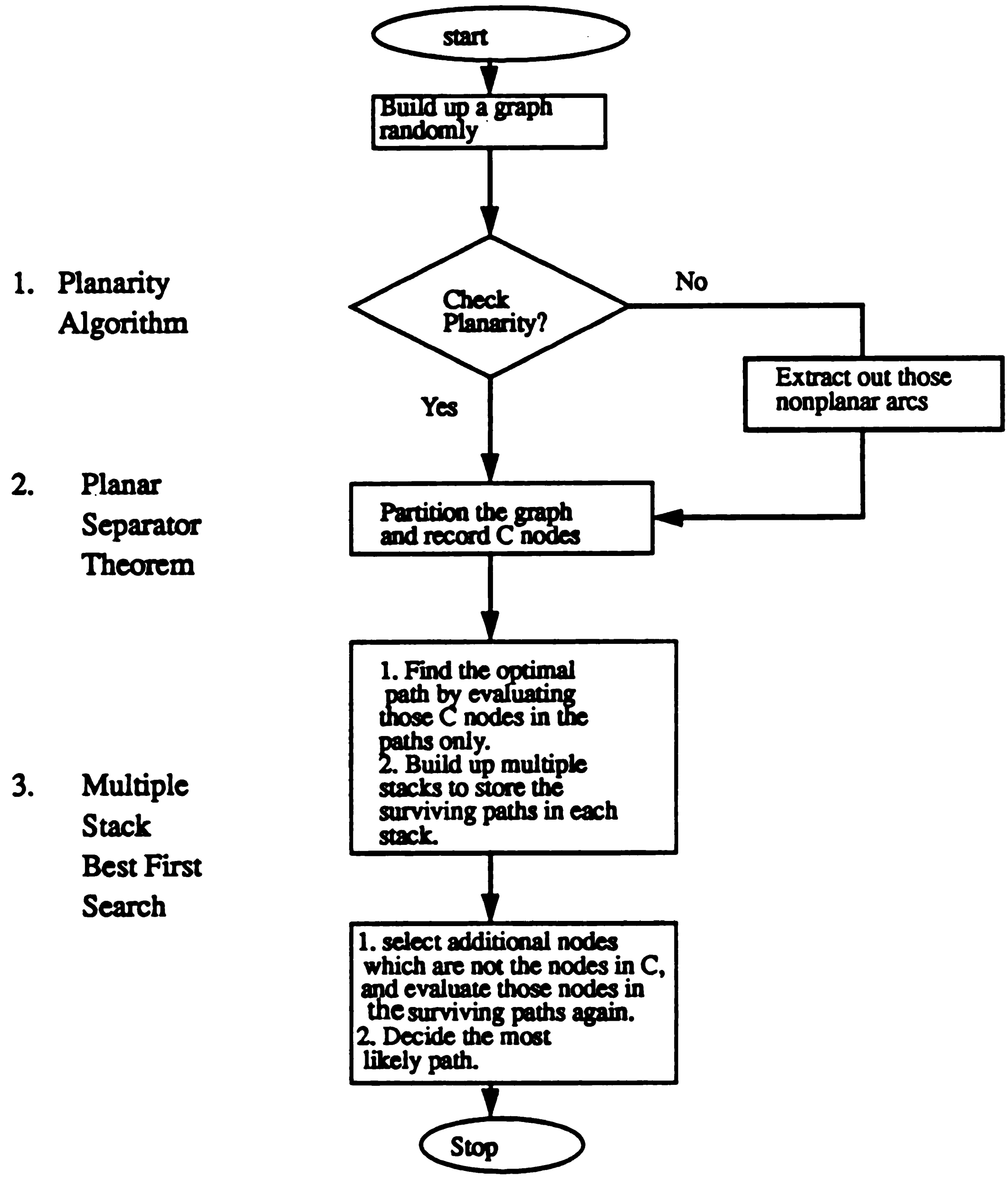


Figure 3: A procedure for graph partitioning approaches in signal decoding.

2 Planarity of a Finite Graph

In this chapter, classical work concerning planar graphs is reviewed, and this material is used to develop a planarity testing algorithm [9] for determining whether or not a given finite graph is planar. If the graph is planar, then a topological embedding is found for future usage; if it is not planar, then some arcs causing nonplanarity are set aside for future reference. Finally, a way to produce a *planar* decoding graph for testing the graph partitioning method is presented.

2.1 Preliminaries

In this section, some basic properties of planar graphs are noted. A graph G(V,E) is called a *planar graph* if it can be drawn, or embedded in the plane in such a way that the edges of the embedding intersect only at the vertices of G(V,E), i.e. no two edges, share any vertices, except at their ends. Fig. 4(a) shows example planar and nonplanar graphs, and in Fig. 4(b) different embeddings of the same graph are shown.

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 can be regarded as a closed walk[5]. A face f is said to be *incident* with the vertices and edges on its boundary. Figure 5 indicates the faces of a particular embedding of the graph. Let $b(f)$ denote the boundary of the face f. For example, the boundary of f_3 in Fig. 5 is as follow:

$$b(f_3) = v_4 e_{4,8} v_8 e_{5,8} v_5 e_{1,5} v_1 e_{1,13} v_1 e_{1,4} v_4$$

[5]A walk in G is a finite non-null sequence composed of some alternately vertices and edges; a closed walk is a walk whose origin and terminus are the same.

Of course, any planar representation of a (finite) graph always contains one face enclosing the graph. This face, called the *exterior face*, is f_1 in Fig. 5. Note that if e is a cut edge[6] in a planar graph, then only one face is incident with e; otherwise, there are two faces incident with e. For instance, $e_{1,13}$ is a cut edge in Fig. 5. Thus, 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_6) incident with it. Later, this property is used to find a proper vertex partitioning.

There are some important properties related to the planarity of a finite graph. These properties are not proved in detail. However, proofs can be found in the references indicated later. The understanding of these properties is the prerequisite to understand the Planar Separator Theorem of Lipton and Tarjan (Theorem 1).

Theorem 2 (Kuratowski's Theorem [12]) *A graph is said to be nonplanar if and only if there is a subgraph of* G *which is* homeomorphic[7] *to either* $K_{3,3}$ *or* K_5.

$K_{3,3}$ and K_5 are called Kuratowski subgraphs, shown in Fig. 6. Neither $K_{3,3}$ nor K_5 is planar. Using Kuratowski's theorem, Lipton and Tarjan show that if any edge of a planar graph G is shrunk to a single vertex, the contracted graph will also be planar [4, Lemma 1]. Furthermore, if G is any planar graph, then shrinking any connected subgraph of G to a single vertex preserves planarity.

Theorem 3 *For any connected planar graph with* $N \geq 3$, *the following holds:*

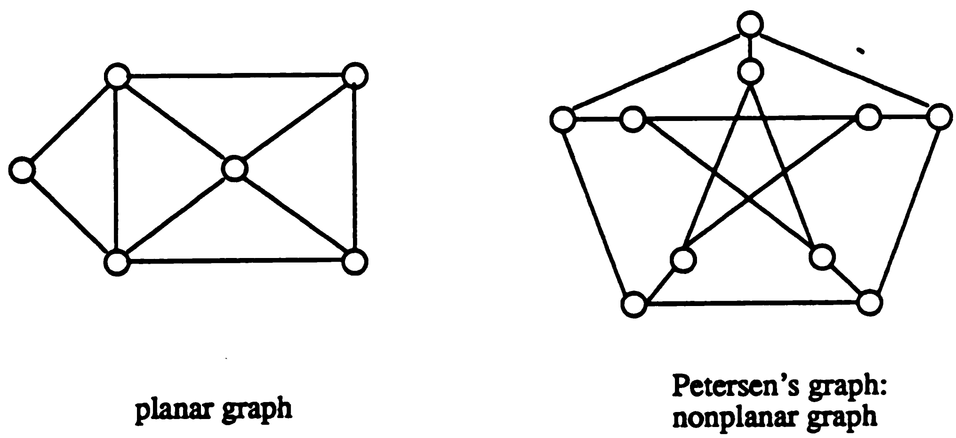
$$|\mathrm{E}| \leq 3N - 6$$

where $|\mathrm{E}|$ *is the total number of edges,* N *is the total number of vertices.*

[6] A *cut edge* in G is an edge whose removal will disconnect G.

[7] Two graphs are said to be *homeomorphic* if both can obtained from the same graph by the insertion of new vertices of degree two, in edges; i.e. an edge is replaced by a path whose intermediate vertices are all new added.

(a)



(b)

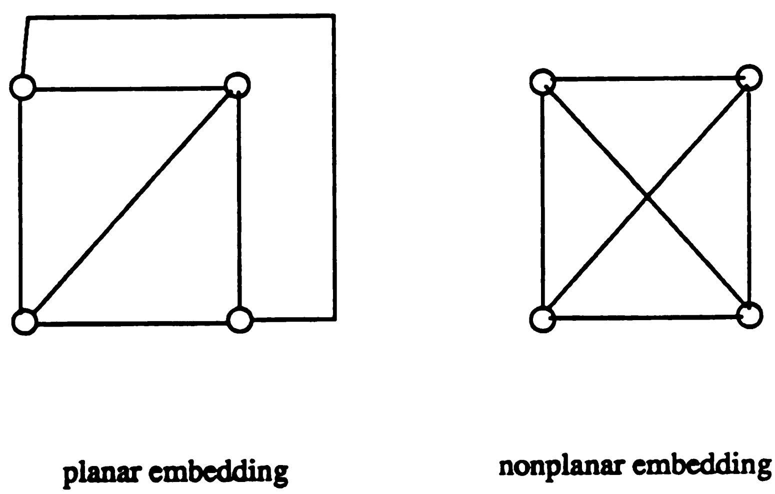


Figure 4: (a) An example of the planar and nonplanar graphs .
(b) The same graph but different embeddings

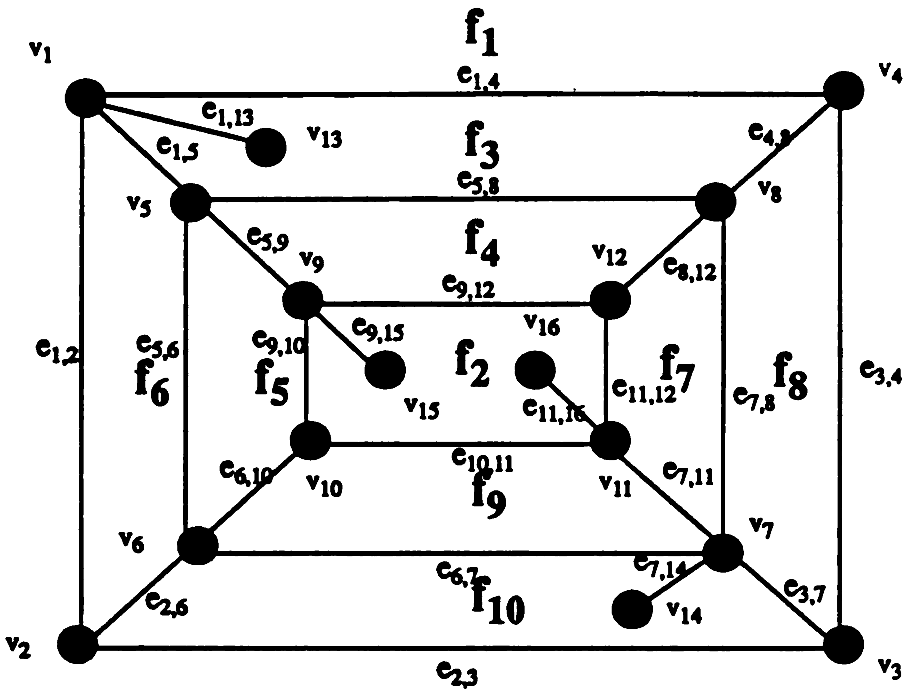


Figure 5: A planar graph with ten faces, where f_1 is called the *exterior face.*

This theorem results from *Euler's formula* , $\phi = |\mathbf{E}| - N + 2$, in which ϕ is the total number of faces of G [11]. Conversely, if $|\mathbf{E}| > 3N - 6$, then the graph is nonplanar.

Theorem 4 (Jordan Curve Theorem [4]) *Let C be any closed curve in a planar graph. The removal of C divides the plane into exactly two connected regions, the inside and the outside of C.*

2.2 A Planarity Testing Algorithm and a Topological Embedding

Kuratowski's Theorem is the earliest characterization of planar graphs. This theorem proves that no planar graph contains either a complete graph on five vertices or a complete bipartite graph on six vertices as shown in Figure 6. Even though Kuratowski's statement is elegant, his condition is not useful as a practical test of planarity. In

(a) (b)

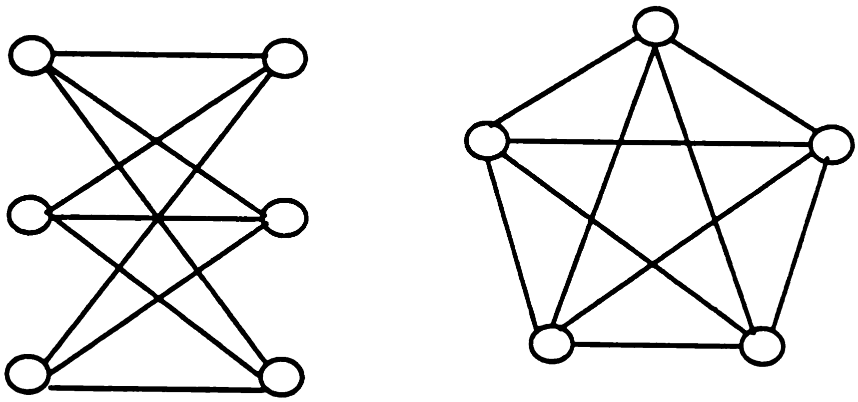


Figure 6: Kuratowski subgraphs (a) $K_{3,3}$; (b) K_5.

this work, to test for planarity, we attempt to construct a planar embedding of the given graph. If such a representation can be completed, then the graph is planar; if not, then the graph is nonplanar.

Hopcroft and Tarjan [7] were first to show that planarity testing can be done in linear time ($\mathcal{O}(N)$). In their algorithm, they also show how to draw the graph if it is planar. The algorithm starts by finding a simple cycle and adding to it one simple path at a time. Each new path connects two old vertices via new edges and vertices. However, the Hopcroft-Tarjan planarity testing algorithm does not give a clear description of how to determine each face of a planar graph. At the same time, this algorithm is fairly complex and a complete description would require a much more elaborate exposition. It might be possible to employ this efficient planarity testing algorithm in future work to enhance the entire structure of our partitioning graph search algorithm. However, here we apply the less efficient, but simpler and still polynomial time, planarity testing algorithm, due to Demoucron, Malgrange and Pertuist in 1964 [9]. Demoucron's algorithm is based on a criterion for determining when a path in a graph can be drawn through a face of a partial planar representation of the graph.

The planarity testing algorithm of Demoucron *et al.* is shown in Fig. 7. Before using this algorithm to determine whether a given graph is planar, some preprocessing 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 (see Theorem 3).

The following definitions will be required: Let $\mathbf{G}_i(\mathbf{V}_i,\mathbf{E}_i)$ be a subgraph of $\mathbf{G}(\mathbf{V},\mathbf{E})$, 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 $\mathbf{V}(\mathbf{B},\mathbf{G}_i)$ 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 $\mathbf{V}(\mathbf{B},\mathbf{G}_i)$ 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 $\mathbf{G}_1,\mathbf{G}_2,\cdots$ of planar subgraphs of $\mathbf{G}$, and corresponding planar embeddings $\mathbf{H}_1,\mathbf{H}_2,\cdots$ 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 Fig. 7. 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 nonplanar. 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*, $|\mathbf{E}| - N + 2$ faces will be found. Since these faces have been found following implementation of the process shown in Fig. 7, a fixed planar embedding of the graph $\mathbf{G}$ can be constructed. In Chapter 3, this result will be used to find an appropriate vertex partitioning.

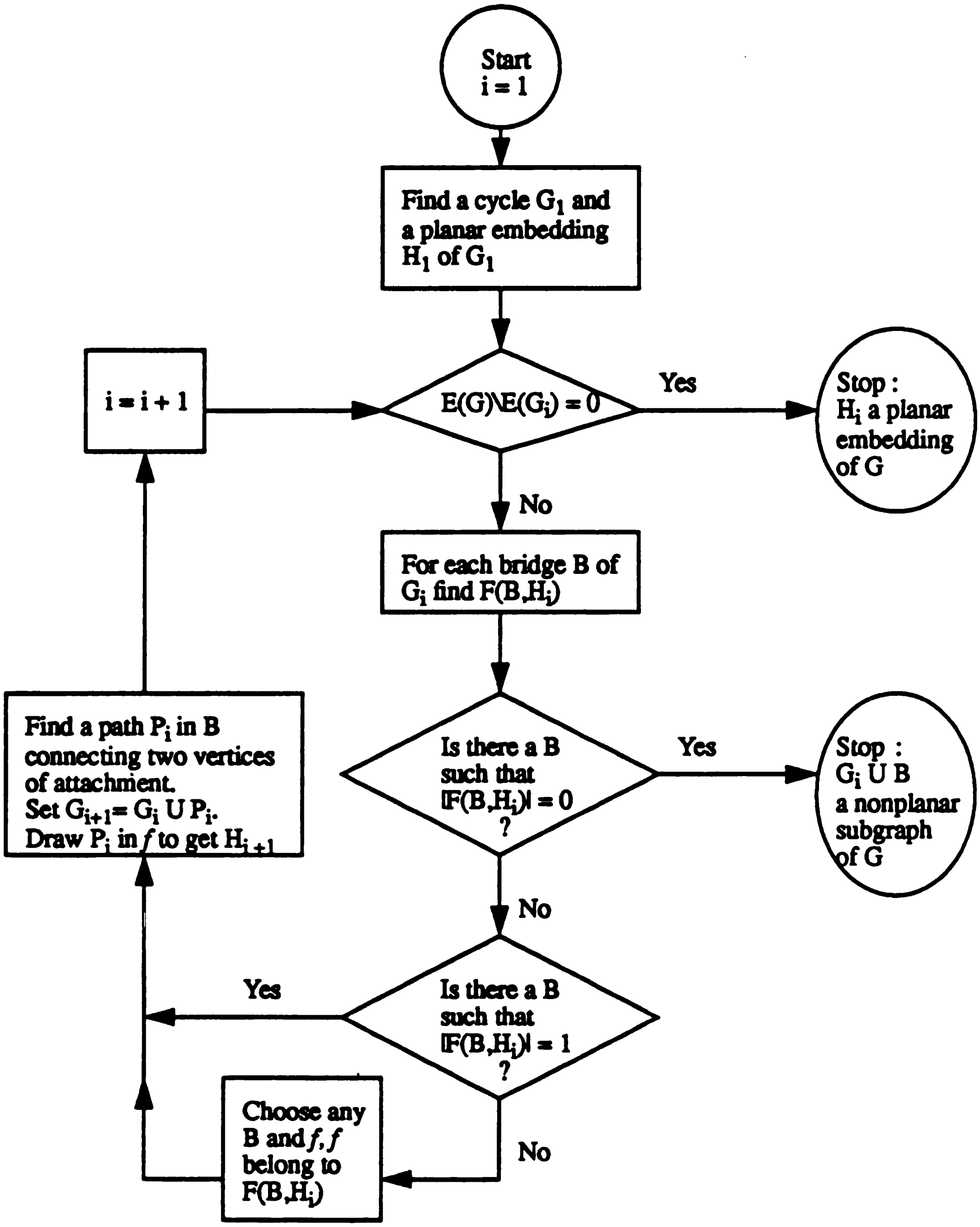


Figure 7: The planarity testing algorithm of Demoucron *et al.* [9]

2.3 Creating a Planar Decoding Graph

A simple ***language graph***, an example of a decoding graph, for experimental testing of the partitioning methods is constructed as follows: Consider a set of sentences, $\mathbf{\Sigma} = \{\Sigma_i\}$, composed of discrete words from the set, $\mathbf{W} = \{w_i\}$. A sentence (of length T) is of the form, $\Sigma_i = w_{i1}, w_{i2}, \ldots, w_{iT}$, where the comma denotes concatenation. Let's assume that the word set and the sentence length are finite, so that the set of sentences can be representable 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, a ***Markov language graph*** $\mathbf{G}(\mathbf{V},\mathbf{A})$ consists of a set of vertices, $\mathbf{V} = \{v_i\}$, a set of edges, $\mathbf{A} = \{a_{kj}\}$ (a_{kj} connects vertex v_k to vertex v_j), a special vertex $s \in \mathbf{V}$ indicating the start vertex of each path, and a set of transition weights, $\{P(v_j \mid v_i)\}$. $P(v_j \mid v_i)$ is the probability that $w(v_i)$ is followed by $w(v_j)$ in any sentence, where $w(v_k)$ denotes the word associated with vertex v_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 v_k) will have one-to-one correspondence to the elements in $\mathbf{\Sigma}$.

An example of a language graph is shown in Figure 8(a). This graph is created from the following list of sentences, each sentence begins with a dummy point, say "•", the dummy point is the starting vertex s of each path in the graph $\mathbf{G}$ (or, equivalently, the first word of each sentence in $\mathbf{\Sigma}$). The sentences are :

- It is a language graph example
- It contains a list of sentences
- He is not very angry
- He wants a piece of paper

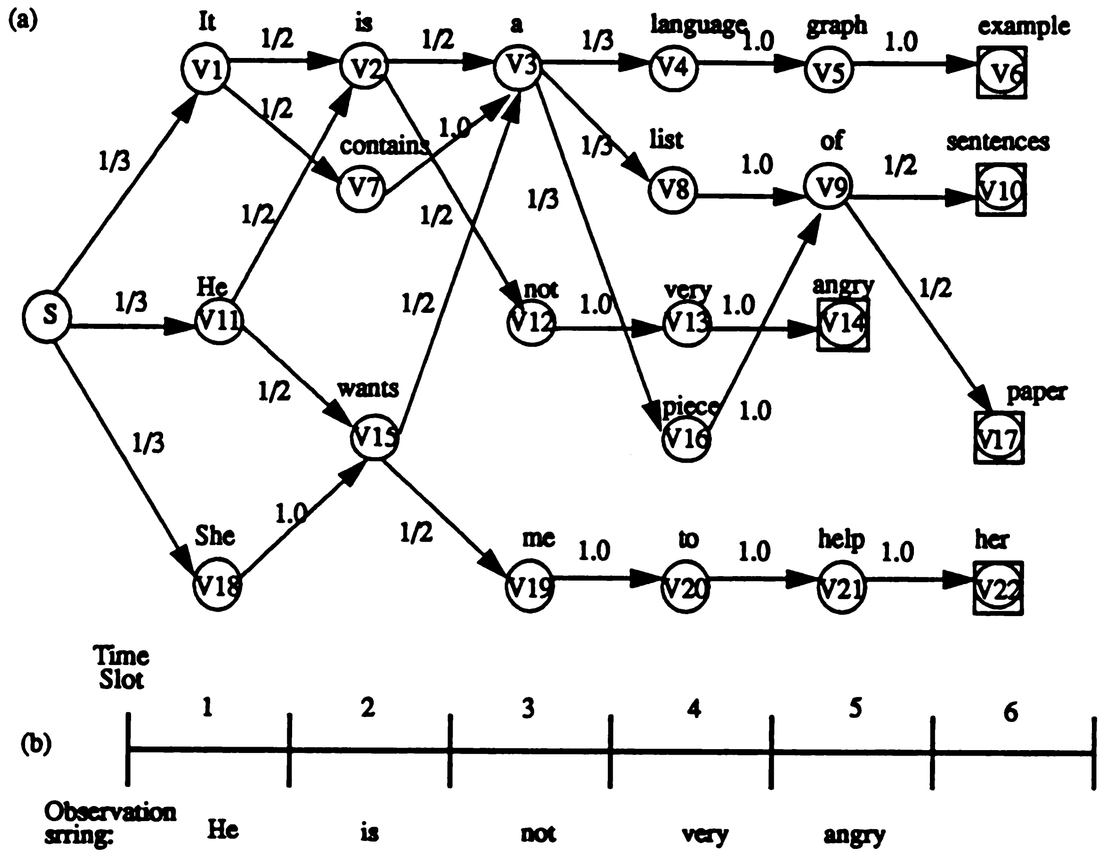


Figure 8: (a)An example of a decoding graph. The transition weights are shown on each edge.
(b)The boundaries of the observation string are known.

- She wants me to help her

The goal is to find a planar subgraph of the language graph by extracting out the planarity breaking arcs (arcs which make the language graph G nonplanar). Note that the planar subgraph, say G′, will 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. Since G(V,E) is planar, G(V,A) will also be planar. The way to discover the planarity breaking arcs in the original graph G is as follows:

A data file to store the set of sentences Σ is created (remember that each sentence begins with a dummy point $\bullet$). The sentences are stored in the data file one by one, and every word in a sentence is stored line by line in the data file according to the concatenation of the words in a sentence. After building up a data file by 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. 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 directed graph is obtained. However, the digraph created might not be planar. Therefore, when one line is read to build a partial graph, the planarity testing algorithm discussed in Section 2.2 is used to determine whether the newly added edge will result in nonplanarity. If so, it is extracted from the partial graph and the next line of the data file is read. Finally, a planar graph is obtained. An example showing the creation of the planar graph using this method is shown in Fig. 9.

Note that when the data are read and checked for planarity line by line, there are two situations which will never make the graph nonplanar. One is the reading of the dummy point $\bullet$, because it is the first word of each sentence, or equivalently, it is the starting vertex of every path in the graph G'. The other is the case in which the entire content of the present line has not been previously entered. After reading in this line, a new path of vertices and edges is added to the partial graph. However, the new path will not cause nonplanarity.

Note also that the planar decoding graph constructed by using this procedure is *connected*. From the discussion of the situations which will never make the graph nonplanar, it can be concluded that the only situation in which the planarity breaking arcs occur is when at least one word of the current line is already resident in some node. The new edge formed by reading in this line which causes nonplanarity is deleted. However, both ends of the edge are "old" nodes in the partial graph. These nodes are still connected to some other nodes in the partial graph created so far. Thus, after reading all data and the construction of a graph by this method, the final graph is a connected planar graph.

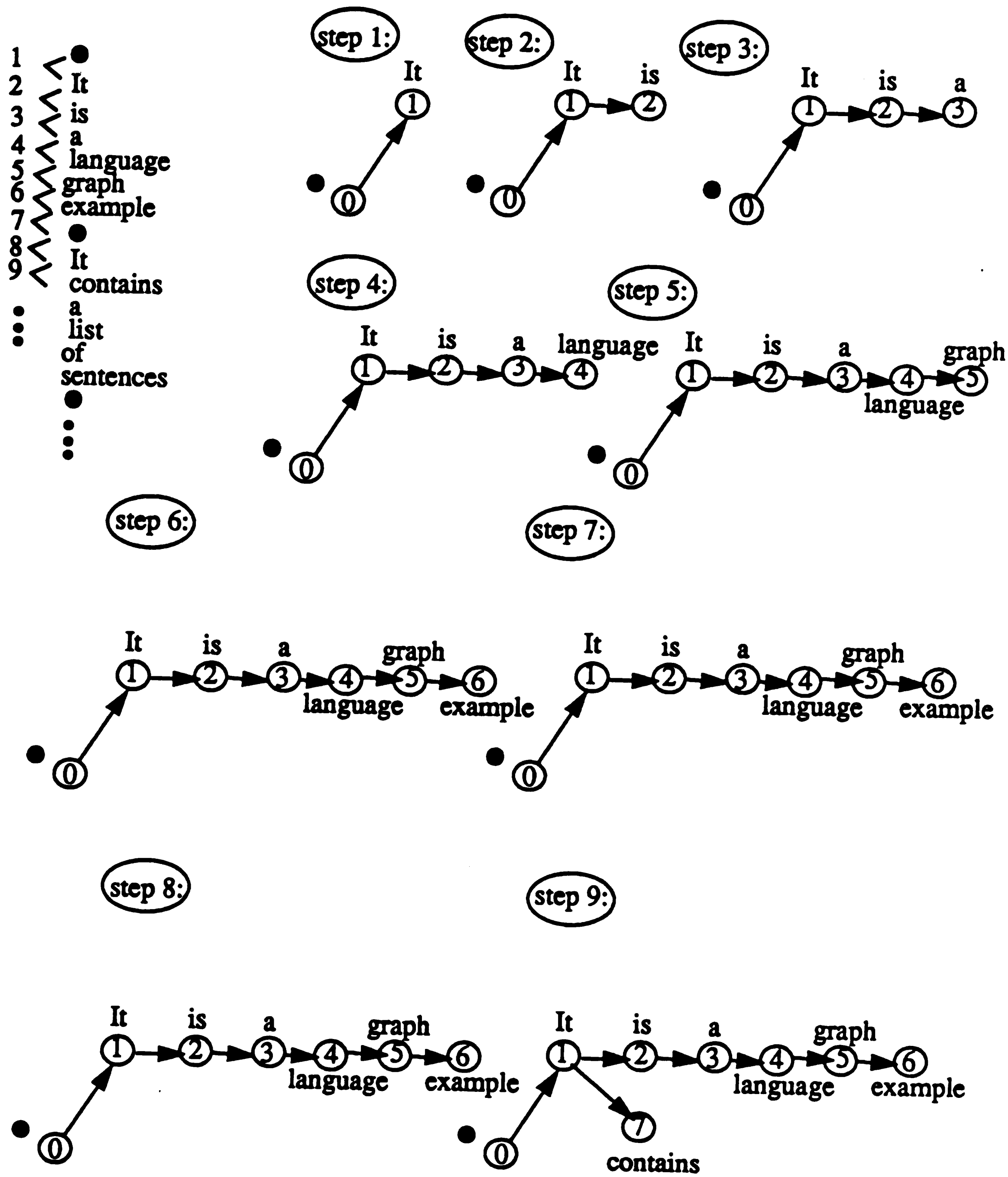


Figure 9: An example of creation of a planar decoding graph.

3 Development of a Partitioning Algorithm

In this chapter, the $\sqrt{N}$-Planar Separator Theorems are discussed and a vertex partitioning algorithm for actual implementation is developed. The results in this chapter are closely related to the effective use of the divide-and-conquer[8] strategy for solving problems on planar graphs.

3.1 Planar Separator Theorems

For problems defined on graphs, there are some general conditions under which the divide-and-conquer approach is useful. Let Ψ be a class of graphs closed under the subgraph relation (i.e., if $G_1 \in \Psi$ and G_2 is a subgraph of G_1, then $G_2 \in \Psi$). In [4], an $f(N)$*separator theorem* for Ψ is a theorem of the following form:

Theorem 5 *There exist constants $\alpha < 1$, $\beta > 0$ such that if G is any N-vertex graph in Ψ, the vertices of G can be partitioned into three sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ such that no edge joins a vertex in $\mathcal{A}$ with a vertex in $\mathcal{B}$, $|\mathcal{A}|$, $|\mathcal{B}| \leq \alpha N$, $|\mathcal{C}| \leq \beta f(N)$.*

In 1979, Lipton and Tarjan [4] proved that a $\sqrt{N}$-separator theorem (see Theorem 1 in Subsection 1.2) holds for the class of all planar graphs with constants $\alpha = \frac{2}{3}$ and $\beta = 2\sqrt{2}$. Djidjev [8] improved the constant $\beta = 2\sqrt{2}$ to $\sqrt{6}$ in 1984. The constructive proof of these separator theorems depends on two fundamental lemmas. Since the algorithm presented here for finding an appropriate vertex partition is based on Djidjev's result, a brief description of Djidjev's separator theorem and the supporting lemmas is necessary.

[8]In [4], the following three conditions are shown to be necessary for the success and efficiency of divide-and-conquer: (i) the subproblems must be of the same type as the original and independent of each other (in a suitable sense); (ii) the cost of combining the subproblem solutions into a solution to the original problem must be small; and (iii) the subproblems must be substantially smaller than the original problem.

Lemma 1 *Let* G *be any N-vertex connected planar graph. Suppose* G *has a spanning tree of radius r. Then the vertices of* G *can be partitioned into three sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, such that no edge joins a vertex in $\mathcal{A}$ with a vertex in $\mathcal{B}$, neither $\mathcal{A}$ nor $\mathcal{B}$ has total number of vertices exceeding $\frac{2N}{3}$, and $\mathcal{C}$ 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 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 (Theorem 4), the cycle divides the graph into two parts, the inside and the outside of the cycle. Lipton and Tarjan [4, Lemma 2] showed by examples that at least one such cycle separates the graph so that neither the inside nor the outside of the cycle contains more than $\frac{2N}{3}$ vertices. Note that the simplest class of graphs with small separators is trees. A tree has the separator $\mathcal{C}$ of size 1, and the root of the tree is a proper separator.

Lemma 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 number of vertices on levels 0 through $l'-1$ does not exceed $\frac{2N}{3}$ and the number of vertices on levels $l''+1$ and above does not exceed $\frac{2N}{3}$, it is possible to find a partition $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ of the vertices of* G *such that no edge joins a vertex in $\mathcal{A}$ with a vertex in $\mathcal{B}$, $|\mathcal{A}|$, $|\mathcal{B}| \leq \frac{2N}{3}$, $|\mathcal{C}| \leq L(l') + L(l'') + \max\{0, 2(l' - l'' - 1)\}$.*

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 $\mathcal{C}$ set, and let $\mathcal{A}$ be all the vertices below the level l' and $\mathcal{B}$ 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 number of vertices between $l' + 1$ and $l'' - 1$ does not exceed $\frac{2N}{3}$. A proper partition is obtained by setting $\mathcal{A}$ the part of the three with the most vertices, $\mathcal{B}$ the remaining two parts, and $\mathcal{C}$ the set of vertices in levels l' and l''. The other case is that in which the total number of vertices between $l' + 1$ and $l'' - 1$ exceeds $\frac{2N}{3}$. In this case, the part between $l' + 1$ and $l'' - 1$ requires sub-partitioning. A sub-partitioning 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. A new graph[10], say G$'$, is formed. Note that the new graph preserves planarity [4, Col 1]. Apply Lemma 1 to the new graph. Let $\mathcal{A}'$, $\mathcal{B}'$, $\mathcal{C}'$ be its vertex partition, the set $\mathcal{C}'$ being the vertices on the cycle. Therefore, a proper vertex partitioning of the graph G derives from letting $\mathcal{A}$ be the set among $\mathcal{A}'$ and $\mathcal{B}'$ with more vertices, $\mathcal{C}$ the vertices in levels l' and l'' plus the set $\mathcal{C}'$, and $\mathcal{B}$ the remaining vertices.

Theorem 6 (Djidjev's Planar Separator Theorem [8]) *Let* G *be any N-vertex planar graph. The vertices of* G *can be partitioned into three sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ such that no edge joins a vertex in $\mathcal{A}$ with a vertex in $\mathcal{B}$, $|\mathcal{A}|$, $|\mathcal{B}| \leq \frac{2N}{3}$, $|\mathcal{C}| \leq \sqrt{6N}$.*

Since our interest is in partitioning the connected decoding graphs for selecting "high payoff" nodes to evaluate, we simply consider the case of connected graphs in this theorem. The partitioning construction involves classifying the vertices of G into

[10]Note that the single vertex x will not be counted in the total number of vertices in G$'$.

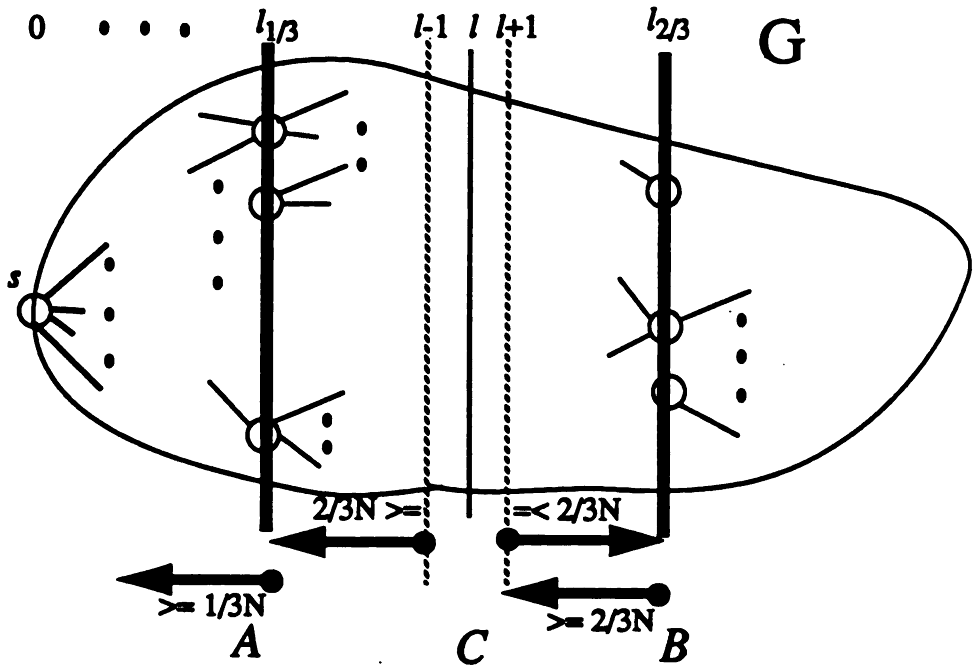


Figure 10: The condition implied by case (1) in Theorem 6. Since we can find the level l satisfying $l \in [l_{1/3}, l_{2/3}]$ and $L(l) \leq \sqrt{6N}$, an appropriate vertex partition is found.

levels according to their distance from some vertex v. Let L(l) be the total number of vertices on level l. Find two levels first, say $l_{1/3}$ and $l_{2/3}$, satisfying certain numerical restrictions: For each $\alpha \in (0,1)$, let l_α denote a level such that $\sum_{l=0}^{l_\alpha-1} L(l) < \alpha N$ and $\sum_{l=0}^{l_\alpha} L(l) \geq \alpha N$. (1) If there exists one level between $l_{1/3}$ and $l_{2/3}$, say l, such that $L(l) \leq \sqrt{6N}$, then the set of vertices on levels 0 through $l-1$ and the set of vertices on levels $l+1$ through above will not exceed $\frac{2N}{3}$. Let C be the set of vertices on level l. Then the theorem is true. This case is shown in Fig. 10. (2) Otherwise, finding a nonnegative integer, say j, satisfying $\sum_{l=l_{1/3}-j+1}^{l_{2/3}+j-1} L(l) < 2/3 \cdot N$

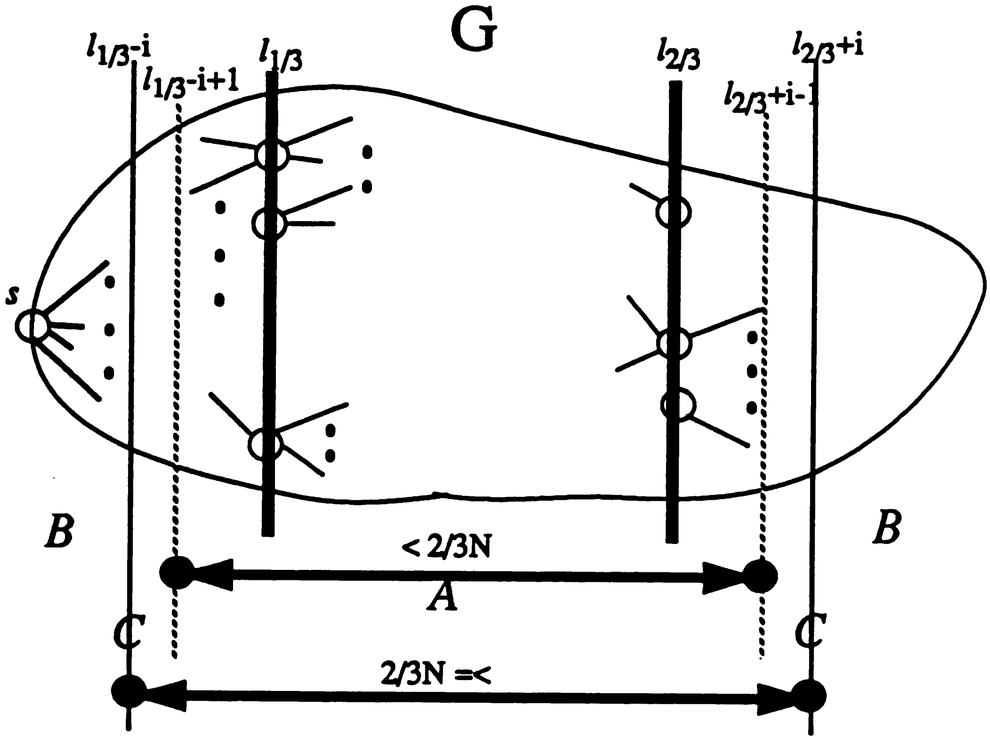


Figure 11: The condition implied by case (2a) in Theorem 6.

and $\sum_{l=l_{1/3}-j}^{l_{2/3}+j} L(l) \geq \frac{2N}{3}$. Two subcases must be considered: (2a) If If there exists i $\in$ [0,j] and $L(l_{1/3}-i)+L(l_{2/3}+i) \leq \sqrt{6N}$, then the nodes in the C set are the vertices on levels $L(l_{1/3}-i)$ and $L(l_{2/3}+i)$. Following the numerical rule for finding j, let $\mathcal{A}$ be the set of vertices on levels $L(l_{1/3}-i+1)$ through $L(l_{2/3}+i-1)$ which does not exceed $\frac{2N}{3}$, and $\mathcal{B}$ the set of the remaining vertices. Thus, this is a proper partition. The case is shown in Fig. 11. (2b) Otherwise, two levels, say l' and l'' satisfying the following numerical restrictions, are located.

$$l' \leq l_{1/3} - j - 1 < l_{2/3} + j + 1 \leq l''$$

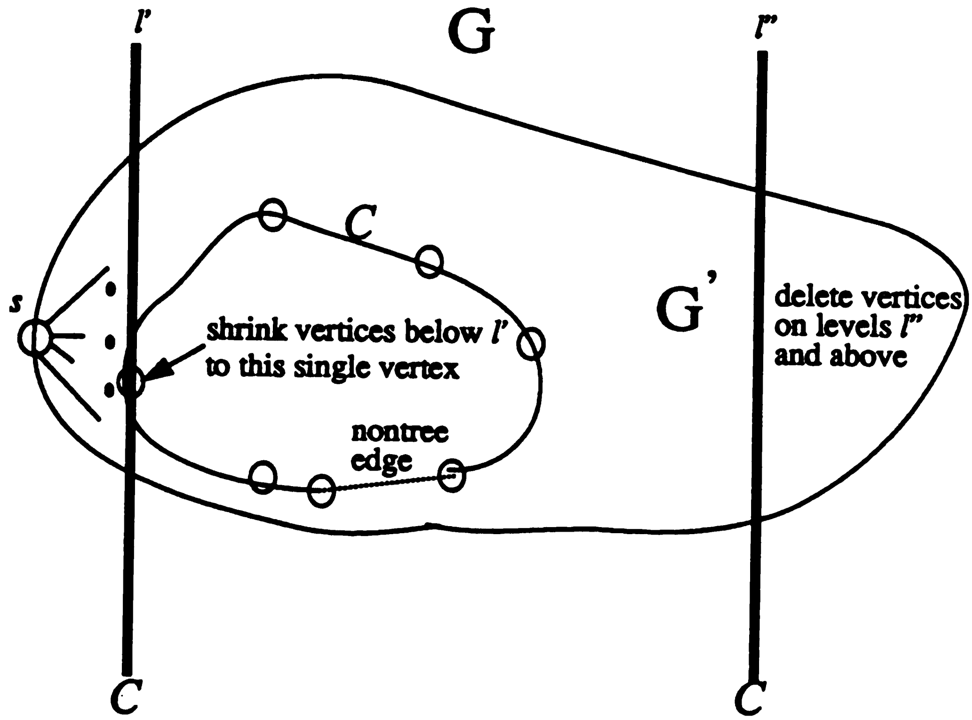


Figure 12: The condition implied by case (2b) in Theorem 6.

$$L(l') + 2(l_{1/3} - j - 1 - l') \leq 2\sqrt{M}$$

$$L(l'') + 2(l'' - (l_{2/3} + j + 1)) \leq 2\sqrt{N - P}$$

where $M = \sum_{l=0}^{l_{1/3}-j-1} L(l)$, $P = \sum_{l=0}^{l_{2/3}+j} L(l)$. Apply **Lemma 1** to the new formed graph between l' and l'', say G$'$, to find a simple cycle separator, and use the result in **Lemma 2** to complete the vertex partitioning of the graph G. Since Djidjev [8] proved mathematically that $L(l') + L(l'') + \max\{0, 2(l' - l'' - 1)\} < \sqrt{6N}$, the set of vertices in C are the vertices in levels l' and l'' plus the simple cycle found in the new graph between l' and l''. The case is shown in Figure 12.

3.2 An Algorithm for Planar Graph Partitioning

The proof of Theorem 6 discussed in Section 3.1 leads to an algorithm for finding a vertex partition satisfying the theorem. The algorithm which has been outlined in Chapter 1 is shown in Fig. 2. The only issue not discussed there is how to find a sub-partitioning of the new graph G', where G' is formed by deleting all the vertices on levels l'' and above, and shrinking all vertices on levels 0 through $l' - 1$ to a single vertex, say x. Note that the single vertex x will not be included in the total number of vertices in G'. Recall that finding the partitioning cycle in G' is necessary only when the subcase 2.b of the proof of Theorem 6 occurs [8].

The algorithm for finding the partitioning cycle in the new graph G' is presented in Fig. 13. Demoucron's planarity algorithm [9] is used to determine the boundary of each face in the planar embedding of G', say H', and to find a breadth-first spanning tree of G' as inputs.

Choose any nontree edge, say (v, w), and form the corresponding cycle by (v, w) and some tree edges in the graph G'. If neither the inside nor the outside of the chosen cycle contains more than $2/3(N' - 1)$ vertices[11], then the cycle will be a proper separator of G'. However, if no set of these nontree edges in the graph G' forms an appropriate cycle, then one new edge is added in a face for each test. The new added edge is a nontree edge which can form a cycle with some tree edges. Note that whenever one new edge is added in a face, this edge will divide the original face into two parts, each part forming a new face of the planar embedding. Each time one 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 contains more than $2/3(N' - 1)$ vertices (see Lemma 1). From the proof

[11] Let N' be the total number of vertices in G' including the single vertex x

the proof of Lemma 2 in [4], at least one such cycle can be found. The problem arises here. How can it be determined which vertices should be considered inside of the cycle and which vertices outside of the cycle? A method which is shown in Figure 13 is proposed.

From Chapter 2, we know that each edge in the cycle has two faces incident to it (note that every edge in a cycle is not a cut edge). We scan each edge in the cycle once, and assign the vertices of the corresponding two incident faces (excluding the vertices on the chosen cycle) to either the inside or the outside of the cycle. Note that these two faces cannot exist on the same side of the cycle. In other words, if one face is located on the inside of the cycle, then the other must be on the outside of the cycle.

Choose one edge in the cycle. Let two incident faces be f' and f''. We denote by two vertex sets $V_{f'}$ and $V_{f''}$ the vertices in the faces f' and f'' excluding the vertices on the chosen cycle. Let F_o be the set of the vertices on the outside of the chosen cycle, and let F_i be the set of the vertices on the inside of the chosen cycle. The rule for determining which face is located on the inside of the cycle and which face is located on the outside is as follows:

1. Assume that f' is located on the inside of the cycle. Check to see if any vertex in $V_{f'}$ is put to the outside of the cycle, i.e. $V_{f'} \cap F_o \neq \emptyset$. If so, f' must be located on the outside of the cycle rather than on the inside of the cycle. Since f' is located on the outside of the cycle, f'' must be located on the inside of the cycle. Set $F_i = F_i \cup \{V_{f''}\}$ and $F_o = F_o \cup \{V_{f'}\}$. Stop and scan another edge in the chosen cycle.

2. Otherwise, check the status of f''. If some vertices in $V_{f''}$ are put to the inside of the cycle, then f'' must be located to the inside of the cycle. Since f'' is located

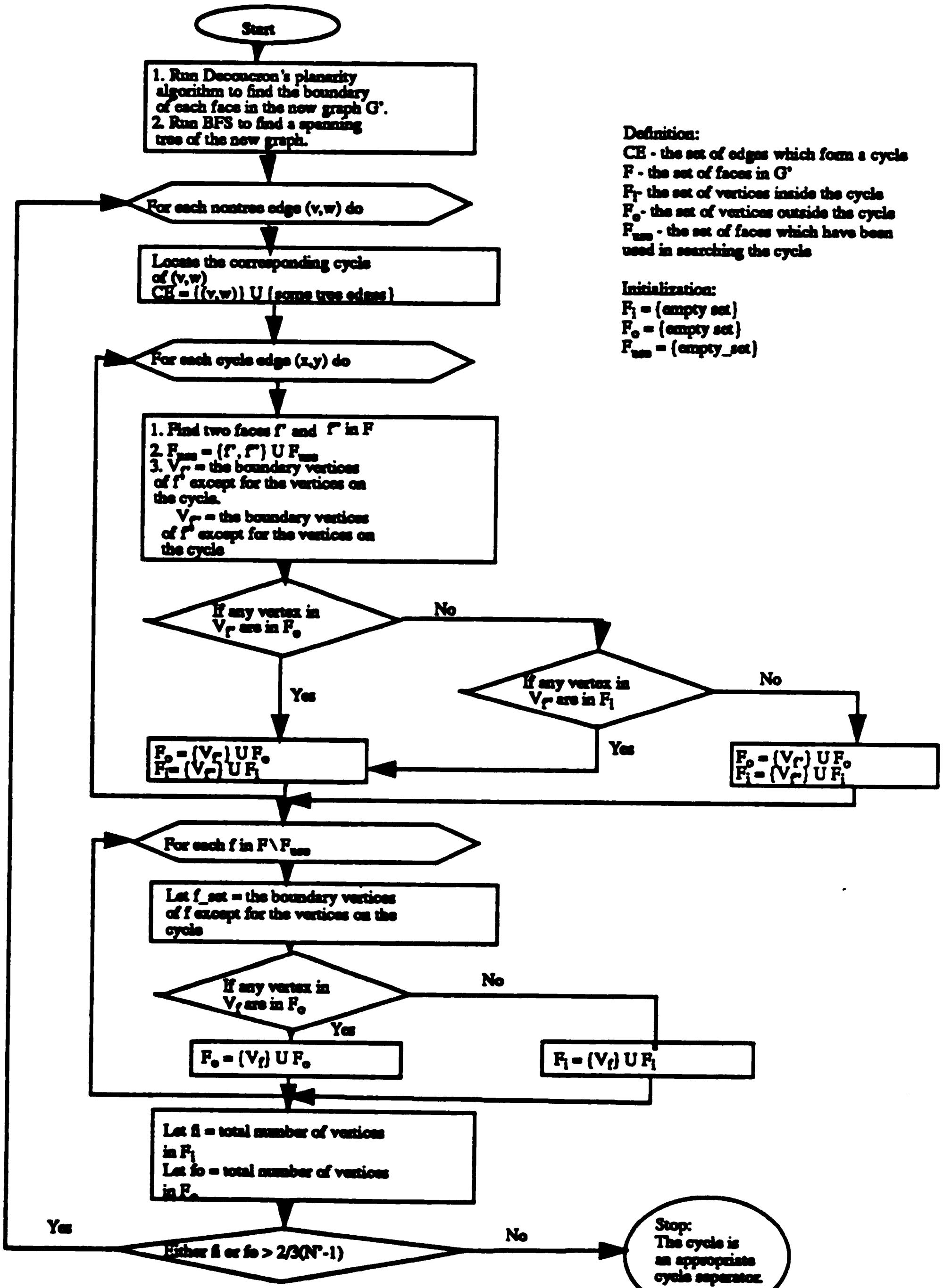


Figure 13: The algorithm for finding the partitioning cycle in the new graph G′.

$F_i = F_i \cup \{V_{f''}\}$ and $F_o = F_o \cup \{V_{f'}\}$. Stop and scan another edge in the chosen cycle.

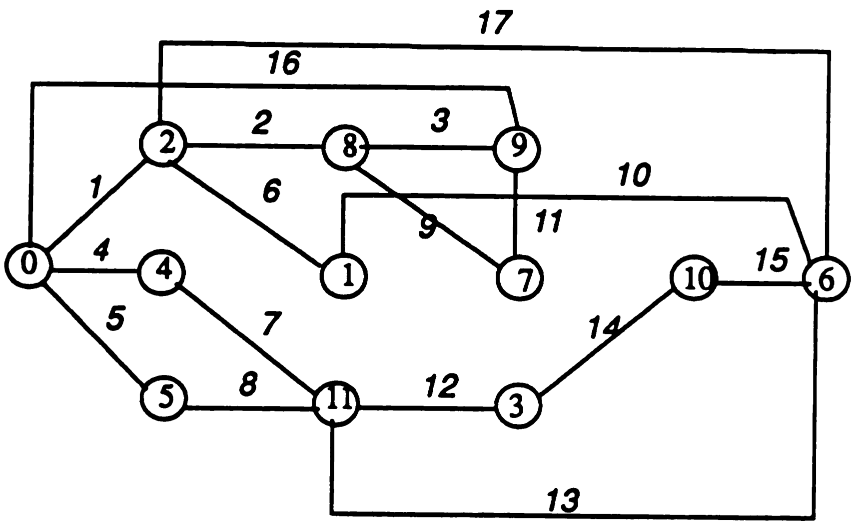
3. Otherwise, put f' to the inside of the cycle and f'' to the outside of the cycle. Set $F_i = F_i \cup \{V_{f'}\}$ and $F_o = F_o \cup \{V_{f''}\}$.

4. Scan another edge in the chosen cycle.

In general, there will be faces which are not incident to any edge in the chosen cycle. If so, the rule described above is used to locate every "unused" face to either the inside or the outside of the cycle. Finally, the vertices on the inside of the cycle and the vertices on the outside of the cycle can be determined.

An example will illustrate the method for finding an appropriate cycle separator. The original graph is shown in Fig. 14. By using Demoucron's planarity algorithm, the boundary of each face is found, and the graph is embedded in the plane. The planar embedding is shown in Fig. 15. The breadth-first spanning tree is shown in Fig. 16. Then one nontree edge, say (3,10), and its corresponding cycle are found. The cycle is shown in Fig. 17. After these processes are completed, three types of data must be recorded. One is the set of edges which forms the cycle. In this example, the edges of the cycle are 1, 17, 15, 14, 12, 7, 4. Another is the set of vertices in each face except the vertices which are on the chosen cycle. This is shown in Table 1. The other is the set of boundary edges for each face. These are shown in Table 2. Now, the rule for determining which faces should be located inside of the cycle and which should be located outside of the cycle is applied. The result is shown in Table 3. From these results, the appropriateness of the chosen cycle as a separator can be determined. This consideration is shown in Table 4.

Finally, we note that the vertex partitioning algorithm presented in this chapter can obtain different partitions of the same graph by choosing a variety of reference



N = 12.
|E| = 17.

Figure 14: The planar graph. Note that every vertex and edge is labeled.

nodes for drawing the level lines. The main criterion in choosing an adequate partition is sufficient path coverage. In the present stage of the research, choosing an adequate partition of the decoding graph G must be done off-line prior to beginning the decoding.

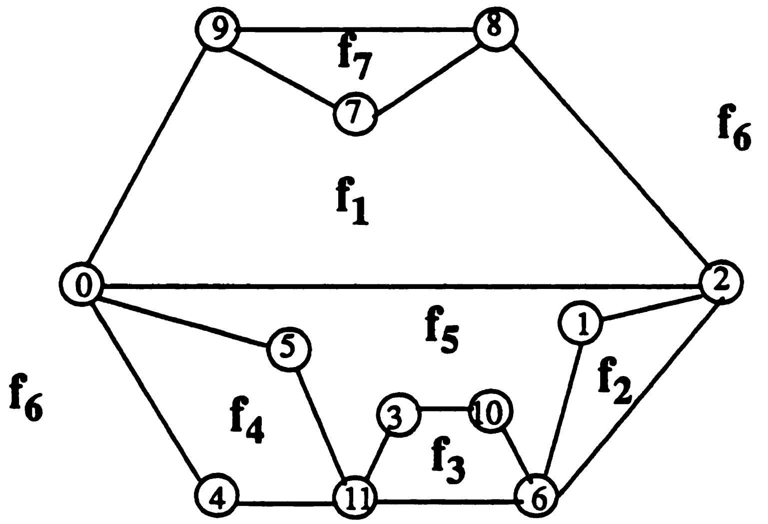


Figure 15: Embed the graph in the plane and find the boundary of each face (Using Demoucron's planarity algorithm).

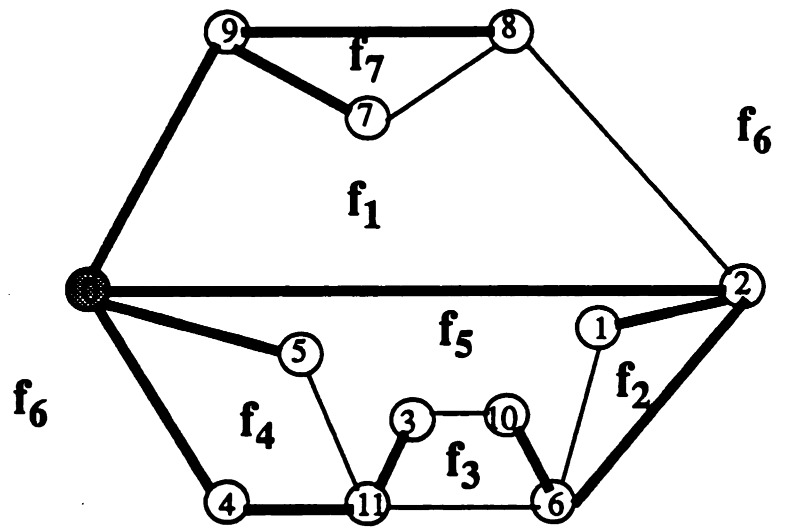


Figure 16: The breadth-first spanning tree of the graph. The bold lines mean tree edges. The node "0" is the root of the tree.

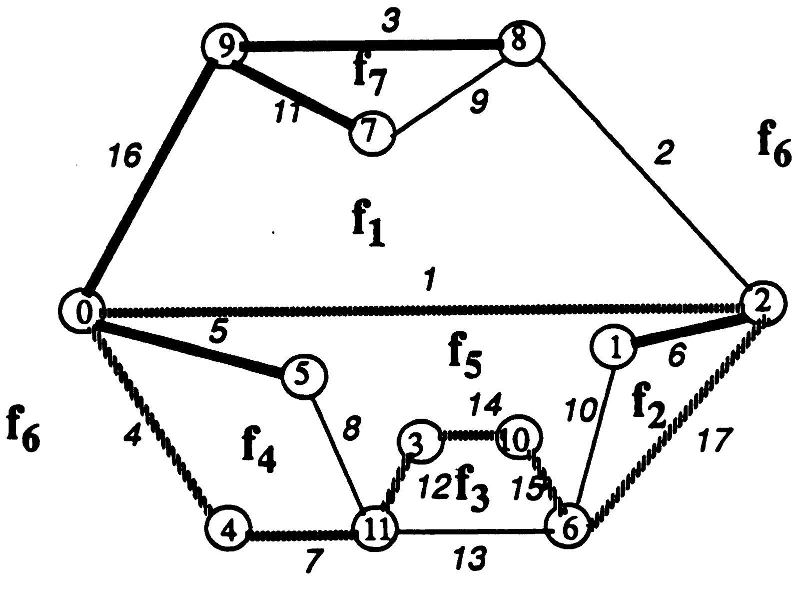


Figure 17: Choose one nontree edge from the planar graph, say (3,10), find its corresponding cycle with some other tree edges. Dotted lines are used to indicate the cycle.

face 1	9, 7, 8
face 2	1
face 3	none
face 4	5
face 5	5, 1
face 6	9, 8
face 7	9, 7, 8

Table 1: The set of vertices in each face except the vertices which are on the chosen cycle.

face 1	16, 11, 9, 2,1
face 2	10, 6,17
face 3	12,13,14.15
face 4	4, 5, 7, 8
face 5	1, 6, 10, 15, 14, 12, 8, 5
face 6	16, 3, 2, 17, 13, 7, 4
face 7	11, 3, 9

Table 2: The boundary edges of each face.

	inside the cycle	outside the cycle
1	face 1	face 5
17	face 6	face 2
15	face 3	face 5
14	face 3	face 5
12	face 3	face 5
7	face 6	face 4
4	face 6	face 4

Result:

Faces have been used to locate in either side of the cycle:
face 1, 2, 3, 4, 5, 6.

Faces haven't been used: face 7.

Faces inside the cycle: face 1, 3, 6, 7.
Faces outside the cycle: face 2, 4, 5.

Table 3: Two faces to which each edge of the cycle is incident. The faces are located to either side of the cycle. The unused faces are recorded.

	Vertices	Total number
Inside	9, 7, 8	3(< 2/3N)
Outside	1, 5	2 (< 2/3 N)

Table 4: The set of vertices on each side of the cycle.

4 Graph Search with Partitioning In Signal Decoding

4.1 Application of the PST to Graph Search Problem

In this section we give a simple example to illustrate the use of the partitioning methods in signal decoding. Since the primary focus of this work has been on the development of partitioning algorithms, this example is not meant to be completely illustrative of the power of the methods, nor does it dwell upon many important details (to be noted) which will be the subjects of future research.

Let's consider the meaning of **G** in the decoding problem. Each node in **G** represents a physical entity in the sense that "resident" at v_i is some abstract information or model which represents the entity. For example, the node might represent a word in a *language graph* (see Section 2.3), and resident at the node would be a statistical model of the word features to be evaluated against measured acoustical observations. Because we are working with a simple speech recognition problem in this example, we will refer to the physical entity which is represented by a node as a *word*. A legal concatenation of words (path through **G**) will be called, naturally, a *sentence*. The evaluation of a node v_i will refer to some quantitative assessment (either likelihood or probability) of the model at v_i with respect to the observations. The decoding problem is to find the most likely path (most likely sentence) in **G**, given the observation string, say **Y**, and the *a priori* structure embedded in **G**.

After using the graph partitioning method to locate $\mathcal{O}(\sqrt{N})$ "high payoff" nodes in an N node decoding graph **G**, the technique for implementing the search of paths must be considered. In conventional "left-to-right" strategies, the evaluation of nodes takes place as they encountered along paths. However, the evaluation of nodes occurs as the *selected* nodes are encountered along paths in the partitioned case. Since we only evaluate the preselected nodes of the paths in the partitioned case, the search procedures must be modified to accommodate unevaluated nodes. The details of this issue are the subject of the next section.

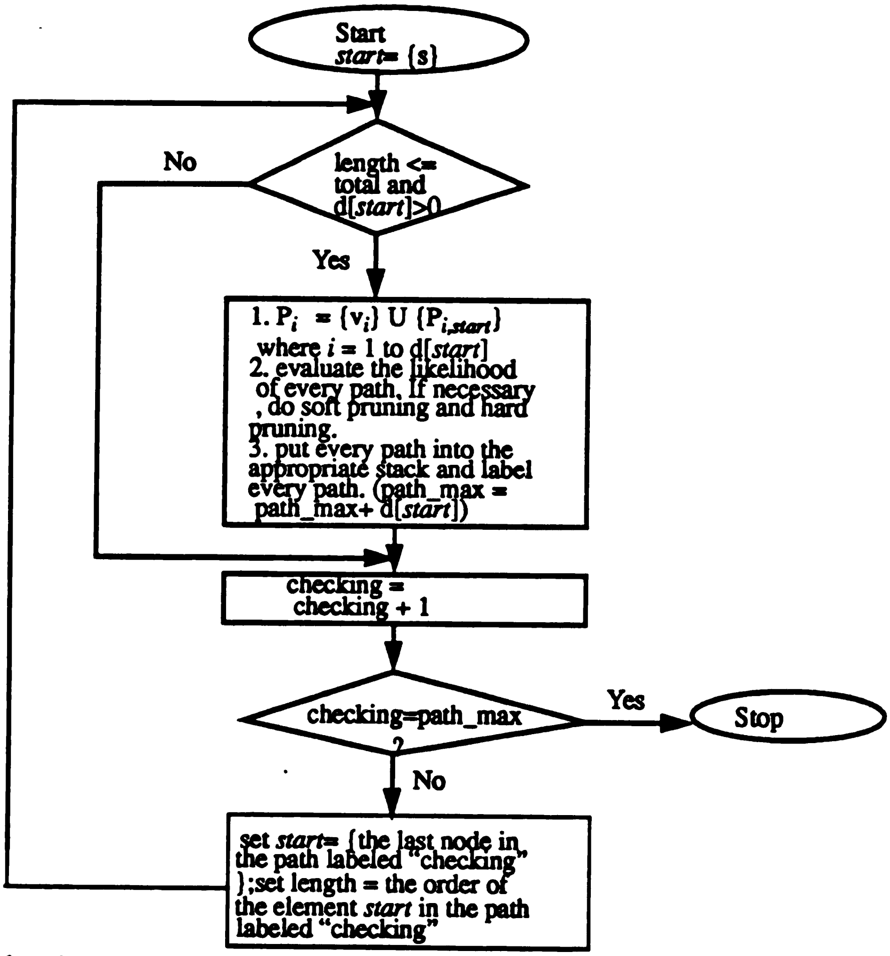
4.2 A Multiple Stack Algorithm for Search with Partitioning

A conventional left-to-right search can be carried out using a "stack" algorithm [2]. Each evolving path is entered into a "stack" (memory array), its position in the stack determined by its likelihood. The most likely partial path is put at the top of the stack. 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* [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 type of pruning is called *soft pruning.* At each iteration, the partial path in the top location of the stack is extended by one word (then paths are rearranged if necessary). When a complete path appears as an entry at the top of the stack, the decoding is complete.

To search the paths in the partitioned case, a modified left-to-right procedure is suggested by Deller in [1]. For simplicity, we consider a special case of this procedure in which the temporal boundaries in the observation string, $\mathbf{Y}$, are known. By this we mean that discrete groups of observations are known to be associated with particular time slots in the utterance and can therefore be associated with exactly one word (node in G). Of course *which* node is unknown, but the known boundaries in $\mathbf{Y}$ greatly simplify the search process. The algorithm shown in Fig. 18 pertains to the graph search in which the boundaries of the observation string, $\mathbf{Y}$, are known. Let's begin generating paths from the leftmost (start) node in $\mathbf{G}$.

The evaluation of nodes occurs as the *selected* nodes are encountered along paths. To provide "fair competition" among partial paths with different numbers of evaluations, a separate stack, say S_i, is built for each number of evaluations. S_i denotes the stack containing partial paths with exactly i evaluated nodes. As the decoding process procedes, path segments will move into the increasingly "more evaluated" stacks as more selected nodes are encountered. Hard pruning and soft pruning can also be applied to the multiple stack graph search. The harding pruning takes place in each stack when there is room for only, say q_i, partial paths in stack S_i. q_i is assigned for each stack prior to search. Soft pruning occurs in stack S_i when a partial path with i evaluations falls below some predetermined likelihood threshold, even though there is sufficient room for it in S_i.

Let $\mathbf{Y}_{t_1,t_2}$ denote the substring of observations t_1 through t_2. Here we take as the likelihood (evaluation) of node x, $P(\mathbf{Y}_{t_1,t_2} \mid x)$ when observations t_1 to t_2 are known to be associable with the time slot in which x is found. Assume that the observation string with known boundaries is of length T. When each path extension in the separate stacks reaches the length T, the question remains as to how to select the optimal path. If the "highest" stack which contains a path is S_I, we need to select a sufficient number of additional nodes in paths of lower stacks to make every surviving path move to the highest stack. What remains in the stacks are partial paths which represent a small subgraph of $\mathbf{G}$. We simply search the subgraph using the standard left-to-right method, the new search problem will be very significantly scaled down with respect to the original problem. Further if I is unacceptably small, further evaluations might be necessary on paths in S_I. The optimal path with the best likelihood is found at the top of the highest stack.



initalization: length = path_max = 0;

checking = -1;

variable : length - total no. of words we search up to now

total - total no. of words in observation seq.

d[] - total no. of nodes adjacent to *start* (*start* - path searching begins)

$P_{i,start}$ - path which has been searched with "*start*" as its last node

P_i - temperary path order i searched from $P_{i,start}$

checking - the label of the path

v_i - the node which is adjacent to *start* in the digraph

Figure 18: A multiple stack decoding algorithm.

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. This procedure is particularly attractive if the partitioning can be done in real time. The solution would be expected to rapidly converge. Each partition and search involves $\mathcal{O}(\sqrt{N})$ or fewer evaluations.

4.3 Application Example

In [1], an example with relatively few nodes was provided to keep the resulting graph visually tractable. However, for the purpose of better illustrating the power of the partitioning graph search method, a large planar graph with 1,061 nodes and 1,196 edges is created in this work for the experiment.

The method presented in Section 2.3 for creating a connected planar graph is applied to construct a large graph. The original data file is built by using the random number generator in a C programming library to generate a few "sentences", each sentence, or equivalently each path in the resulting digraph, consists of $33-40$ words (nodes). Every word is represented by an integer number created from the random number generator. Note that different integer numbers represent different words in the sentences. The data file is shown in Appendix A. The resulting digraph (after extracting 119 planarity breaking arcs), say G, is composed of 1,061 nodes (including one dummy node $\bullet$) and 1,196 edges. The nodes of the graph G are shown in Appendix B, and the planarity breaking arcs extracted from the original data file (or, the original graph) are shown in Appendix C.

After creating the planar digraph, the next task is to apply the partitioning algorithm to its underlying undirected graph. The main purpose of the partitioning algorithm is to partition the vertices of the planar graph and find the nodes in the $\mathcal{C}$ set for evaluation. Therefore, the direction of each edge in the decoding graph G need not to be considered as the partitioning process procedes. The node "0" is chosen to be the reference node for classifying the vertices of G into levels. In this graph, the vertices are partitioned into 58 levels from level 0 to level 57. Using the partitioning algorithm, there are a total of 63 nodes in the $\mathcal{C}$ set. These nodes are selected in this graph for evaluation. The position of these selected nodes is shown in Appendix A.

Since these "high payoff" nodes have been chosen, let's further consider how to execute the decoding process. The experiment was carried out as follows:

1. A complete path in the graph was chosen as the given word string. Of course the word string will be unknown in practice. A path containing 37 nodes was selected to be the symbol string.

2. In order to obviate the construction of 1,061 word models (since we knew that only $\mathcal{O}(\sqrt{N})$ of them would be used in the search), we trained 81 models. These trained model are shown in Appendix D. These models corresponded to the $\mathcal{C}$ set nodes found by partitioning, plus a few additional ones (see Appendix A) for a purpose described below. For convenience, the trained models were evaluated in advance with respect to each discrete set of observations representing words in the chosen sentence (correct path).

3. The multiple stack graph search algorithm was used to search for the optimal path. At the end of the search, there were nine stacks created holding 264 candidate paths. This means that the maximum number of the nodes evaluated on a path was nine. The threshold at stack i was given by i times the best match score for any given word in the 81 test words to its correct model. After the threshold was set for pruning the unlikely partial paths at each stack, only 38 paths were remained in the stacks (see Appendix E).

4. A second subset of the key nodes from the surviving paths was evaluated (the left-to-right search approach was applied) to move the surviving paths in the lower stacks to Stack 9. An additional node was added to a path by inserting the appropriate model from the the set of 81 trained model. It was necessary to evaluate 18 additional nodes to move the paths in the lower stacks to Stack

9. The path which had the best likelihood in Stack 9 was the optimal path.

To find a optimal path using the graph partitioning and search method in this graph, 81 nodes are selected for evaluation. Using the conventional left-to-right search to this graph, to evaluate 9 nodes on every path requires 299 node evaluations. The experiment demonstrates that a significant reduction in the number of evaluations is possible with the partitioning procedure with respect to the left-to-right method. In real problems in which a more carefully planned search strategy can be employed, much more improvement than was achieved in this simple example is to be expected. Further, since the partitioning algorithm's main benefit is in reducing the complexity to $\mathcal{O}(\sqrt{N})$, results will be more significant for very large N. N values of $10^6 - 10^{11}$ nodes are not uncommon, for example, in speech recognition graphs [3], [13].

5 Conclusions and Future Work

For signal decoding problems, the graph partitioning method offers a systematic way of locating a very small number of nodes which are guaranteed to give effective coverage of a decoding graph. Through the evaluations of this relatively small number of selected nodes, an optimal path for a given observation string can be found.

The major contribution of this research is the development of a planar graph partitioning algorithm, which can be used to select $\mathcal{O}(\sqrt{N})$ nodes for evaluation from an N node planar decoding graph. In the process, a method for finding an appropriate simple cycle separator to complete the vertex partitioning has been developed. When the search is combined with partitioning, the overall graph search complexity is $\mathcal{O}(\sqrt{N})$. This represents a steep decrease with respect to conventional left-to-right decoding approaches which are usually $\mathcal{O}(N)$. A procedure for circumventing the apparent limitation of these $\sqrt{N}$-Planar Separator Theorems to *planar* graphs is found in [1]. To develop appropriate partitioning algorithms for the more general case will be the subject of future research.

Following partitioning, "scattered" nodes are evaluated and the graph is searched and pruned according to a likelihood measure. To provide "fair competition" among the partial paths with different numbers of evaluations requires the use of an unconventional search algorithm since most existing methods assume the sequential evaluation of nodes from left-to-right. For this purpose, a multiple stack decoding algorithm has been applied to carry out this procedure. In the present research, graph search in the presence of known boundaries in the observation string has been presented. An important issue for future research is the implementation of methods for search with unknown boundaries which is suggested in [1]. Another important issue is the ability to perform in real-time the repartition of the subgraph following a given search.

A summary of future work is as follows:

1. An appropriate vertex partitioning algorithm for generally nonplanar graphs will be developed.

2. Strategies for "optimal" multiple partitioning, and recursive partitioning and search, in real time will be developed and evaluated.

3. Graph partitioning and search algorithms will be applied to the problem of *continuous* speech recognition. The algorithm developed for this search must be applicable to the case of unknown acoustic boundaries.

References

[1] C.G. Venkatesh, J.R. Deller, Jr., and M.B. Cozzens, "A Graph Partitioning Approach to Signal Decoding," in press *Discrete Applied Mathematics* (special issue on Graph Theory in Electrical Engineering).

[2] L.R. Bahl and F. Jelinek, "Decoding for Channels with Insertions, Deletions and Substitutions with Applications to Speech Recognition," *IEEE Transaction on Information Theory,* Vol. IT-21, No.4, pp.404–411, July 1975.

[3] L.R. Bahl, F. Jelinek and R.L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-5, No.2, pp. 179–190, March 1983.

[4] R.J Lipton and R.E. Tarjan, "A Separator Theorem For Planar Graphs," *SIAM J. Computing,* Vol. 36, No. 2, pp. 177–189, April 1979.

[5] G.L. Miller, "Finding small simple cycle separator for 2–connected planar graphs," *Proceedings of the Sixteenth Annual ACM Symposium On Theory of Computing,* pp. 376–382, April 1984.

[6] H. Gazit and G.L. Miller, "A Parallel Algorithm for Finding a Separator in Planar Graphs," *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science,* pp. 238–248, 1987.

[7] J. Hopcroft and R.E. Tarjan, "Efficient Planarity Testing," *J. Assoc. Comput. Mach.*, Vol. 21, pp. 549–568, 1974.

[8] H.N. Djidjev, "On the problem of partitioning planar graphs," *SIAM J. Alg. Discrete Math.,* Vol 3, No. 2, pp. 229–240, June 1982.

[9] J.A. Bondy and U.S.R Murty, *Graph Theory with Applications,* New York: American Elsevier Publishing, 1976.

[10] S. Even, *Graph Algorithms,* Potomac MD: Computer Science Press, 1979.

[11] A. Gibbons, *Algorithmic Graph Theory,* New York: Cambridge University Press, 1985.

[12] J.A. McHugh, *Algorithmic Graph Theory,* Englewood Cliffs NJ: Prentice-Hall, 1990.

[13] 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 NJ: Prentice-Hall, 1980.

[14] S. Rao, "Finding Near Optimal Separators in Planar Graphs," *Proc. 28th Annual IEEE Symposium on Foundations of Computer Science,* pp. 225–237, 1987.

[15] R.J. Lipton and R.E. Tarjan, "Applications of a Planar Separator Theorem," *Proc. 18th Annual IEEE Symposium on Foundations of Computer Science,* pp. 162–170, 1977.

[16] J.R. Gilbert, J.P. Hutchinson and R.E. Tarjan, "A Separator Theorem for Graphs of Bounded Genus," *Journal of Algorithms,* Vol. 5, pp. 391–407, 1984.

[17] L.R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE,* vol. 77, No. 2, pp. 257–285, 1989.

[18] 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.

APPENDIX A

.	663	940	448	1315 <-- C
914	1759	1208	1252	1572
827	332	571	1900	1851
302	1455	1579	20	1944
1631	1685	821	1618	1798 <-- C
785	1716 <-- C	963 <-- C	1630	1094
230	1136	724	272	250
11	1720	762	522	537
1567	1832	1187	947	.
350	751	645	1389	1420
1307	1681	86	1068	434
1339	1106	551	590	1810
929	379	329	476	1655
1216	1719	1018	1634	1820
479	381	975	.	739
703	919	608	267	984
699	1163	964	1140	6
90	219	1157 <-- C	822	1273
440 <-- C	639	1572	629	214
1926	1261	366 <-- C	225	212
1032	40	1377	891	1425
1329	1144	252	1954	1074 <-- C
1682	1941	.	387	1043
1764	1872	1317	1723 <-- C	99 <-- C
1615 <-- C	1569	764	818	882
1961	972	545	1545	1263
1273	1364	1291	641	546
1275	1684	735	234	1984
38 <-- C	931	214	1385	663
923	423	1336	929	626
540	1927	1439	977	1012
1443	1594	1154 <-- C	1834	330
1837	182	544	181	551
1368	.	362	878	1215
1746	1401	1085	206	626
1469	1868	1717	1800	701
505	680	1325 <-- C	508	310
.	538	161	479	876
1480	1940	831	674	1238
424	512	512	1807	1127
678	1289	806	220	297 <-- C
1139	1621	918	94 <-- C	24
1763	1970 <-- C	1416 <-- C	750	1290
1959	1668	1488 <-- C	1048	304
707	1693	1936	80	196
242	352 <-- C	391 <-- C	1028	381

Next page

1288
202
1654
.
415
1079
929
1458
1531
163
1074 <-- C
77
147
1737
1055
1160 <-- C
68
1606
375
1046
659
685 <-- C
1923
249
165
572
274
1807
1228
470 <-- C
188
868
673
1843
723
1088
922
1653
546
453
1816
1972 <-- C
.
1964
1710
290

1124
130
248
1851
1176
908 <-- C
889
1451
1157 <-- C
1406
375
1431
1213
1603
254
1401
824
927
1244
1547
367
519
1200
1265 <-- C
1324
1369
1238
207
1333
1300
849
809
.
1097
1012
1310
357
1901
1114
1515
1307
1489
1298
520
1445 <-- C
1904

274
269
1183
1518
168
1550
389
1721
1168
1714
1090
758
273
775
58
1122 <-- C
1936
192
571
1300
1502
929
.
616
796
861
458
446
1734
255
351
8 <-- C
876
1534
1878
1044 <-- C
1437
268
1117
605
1982
559
1715
255
1686
125 <-- C

1729
1622
317
300
923
1819
1581
477
788
377
1690
1598
.
1424
1853
1527
1784
729
1061
1663
125 <-- C
850
283
1595
1807
265
154 <-- C
1522
872
1841
1647
601
1815
1964
901
1090
136
483
1919
1276
1212
1610
874
36
1034
1079

1915
819
160
977
.
637
179
1117
232
1987
1734
739
1509
606
580 <-- C
1509
1207
747 <-- C
1825
460
1838
313
1295
1757
1589
508
1367
815
896 <-- C
754
246
812
1925
758
141
759
1396
320
228
1980
307
1962
.
169
568
1651

Next page

1678
127
399
1855
587
237 <-- C
169
1883
346
1758
391 <-- C
1714
926
1639
468
1172
451
745
283
592
1504
1679
913
1732
1659
1572
1694 <-- C
731
1741
614
382
1771
741
781
1627
1680
1018
1796
1563
.
1906
306
1431
832
1946
251

357
397
996
640
1342
852 <-- C
671
255
584
682 <-- C
179
630
1413
1921
1596
1796
44
337
577
1671
17
1948
1819
1933
1665
1726
591
1448
910
537
1699
1619
.
1047
611
629
1899
1634
1236
835
317
1415
1817
1730
1688
1413

1878
85 <-- C
102
456
108
471
756
1928
756
773
6
1348 <-- C
221
1268
1885
272
1240
1524
1319
1851
505
1570
1486
1741
757
155
1509
.
1885
1549
339
1764
1634
793
572
95
1264
1680
23
21
453
381
1721
674
1649
1958

946 <-- C
1241
1483
617
1093
1988
187
931
82
944
1086
1943
1870
1323
1492
209
1439
1479 <-- C
1354
.
1926
618
43
1949
991
496
682 <-- C
1064
1522
683
1023
820
1925
506
1437
1370
846
1976
301
928
1272
1739
1223
1142
1062
1068

1703
502
547
1057
865
473
28
909
.
1371
1757
1808
436
1280
491
1811
100
416
317
1890
138
1515
218
439 <-- C
796
1491
178
19
985
1593
1087
1041 <-- C
447
1986
450
1312
811
830
573
1585
202
331
1745
990
1611
237 <-- C

Next page

.
63
1005
1470
305
1144
1337
524
1935
133
367
466
505
1704
59
1592
745
506
1579
1196
170 <-- C
742
378
744
680
580 <-- C
1075
425
1922
1038
1014
1075
1453
20
897
.
1516
235
635
1451
368
1002
1917
873
706
328

818
1452 <-- C
1186
749
1000
1357
1491
1378
101
171
311
1176
597 <-- C
585
566
1963
1661
19
1983
558
1778
1499 <-- C
793
413
1303
1514
1767
1572
739
474
1901
1557
.
1287
658
1030
786
150
1360
897
673
92
25
1622
609
991

1586
622
1363
1569
1180
1493
1421
326
259
1076
192
26
2000 <-- C
931
852 <-- C
901
841
1482
341
1499 <-- C
1112
1489
649
825
3387 <-- C
675
848
83
.
2914
8277
3062
3631
7845
2380
8011
1567
2350
5307 <-- C
3339
8929
9216
6479
4703
6999
9000

4480
3926
1032
3329
3682
5764
21615
7961
9273
31275
4038
4923
5490
7443
7837
41368
7746
61469
8505
.
9480
6424
6678
81139
9763
31959
6707
6242
6663 <-- C
3759
6332
3455
7685
3716
3136
7720
5832
4751
5681
5106
2379
9719
6381
2919
7163
4219

6389
61261
2040
9144
21941
7872
5569
4972
5364
11684
6931
8423
7927
3594
2182
.
3401
9868
6820
6538
3940
6512
91289
9621
7970 <-- C
3668
5693
4352
2940
9208
8571
3579
6821
6963
2724
8762
51187
4645
8600
6551
6329
7018
4975
6080
6964
7157

Next page

.
63
1005
1470
305
1144
1337
524
1935
133
367
466
505
1704
59
1592
745
506
1579
1196
170 <-- C
742
378
744
680
580 <-- C
1075
425
1922
1038
1014
1075
1453
20
897
.
1516
235
635
1451
368
1002
1917
873
706
328

818
1452 <-- C
1186
749
1000
1357
1491
1378
101
171
311
1176
597 <-- C
585
566
1963
1661
19
1983
558
1778
1499 <-- C
793
413
1303
1514
1767
1572
739
474
1901
1557
.
1287
658
1030
786
150
1360
897
673
92
25
1622
609
991

1586
622
1363
1569
1180
1493
1421
326
259
1076
192
26
2000 <-- C
931
852 <-- C
901
841
1482
341
1499 <-- C
1112
1489
649
825
3387 <-- C
675
848
83
.
2914
8277
3062
3631
7845
2380
8011
1567
2350
5307 <-- C
3339
8929
9216
6479
4703
6999
9000

4480
3926
1032
3329
3682
5764
21615
7961
9273
31275
4038
4923
5490
7443
7837
41368
7746
61469
8505
.
9480
6424
6678
81139
9763
31959
6707
6242
6663 <-- C
3759
6332
3455
7685
3716
3136
7720
5832
4751
5681
5106
2379
9719
6381
2919
7163
4219

6389
61261
2040
9144
21941
7872
5569
4972
5364
11684
6931
8423
7927
3594
2182
.
3401
9868
6820
6538
3940
6512
91289
9621
7970 <-- C
3668
5693
4352
2940
9208
8571
3579
6821
6963
2724
8762
51187
4645
8600
6551
6329
7018
4975
6080
6964
7157

Next page

7572	8629	4214	6375	91244
3606	8225	4212	5046	81547
3377	4891	41425	8659	8367
8252	3954	3074	6685	6519
.	8387	7043	7923	7200
7317	7723 <-- C	2099	2249	7265
4764	6818	6882	4165	9324
4545	9545	31263	5792	71369
3291	6410	5546	6274	7238
2735	6234	7984	5807	7207
8214	3385	8663	9228	9333
5336	6929	8626	6470	3300
3439	6977	9012	2188	6849
71154 <-- C	7834	6330	2868	2809
4544	8190	6551	2673	.
8362	4878	7215	9843	3097
1085	6206	2626	6723	9012
7717	3800	2701	9088	5310
5325	2508	3310	8922	4357
41617	6479	2876	5653	21901
2831	6740	7238	2546	3114
6512	3807	21127	6453	11515
4806	4220	6297	7816	5307 <-- C
2918	2094	4024	9972	1489
9416 <-- C	4750	5290	.	71298
7488	41048	2304	7964	4520
9936	8080	4196	9710	5445 <-- C
4391	21028	6381	7290	7904
4448	81315	5288	7124	2274
3252	71572	2202	2130	2689
7900	61851	7654	6248	7183
2020	9944	.	9851	3518
3618	9798	6415	7176	8168
7630	3094	9079	4908 <-- C	5550
2762	2500	8929	2889	6389
5222	4537	3458	21451	61721
4947	.	7531	7157	9168
91389	3420	2163	3406	5714
5068	4434	51074	8375	3090
4590	7810	8077	3431	2758
4760	9655	41497	9213	2273
9634	3820	9737	7603	8775
.	4739	3055	2594	6058
2697	2984	9160	11401	9122
7140	8006	6068	4824	7936
8202	51273 <-- C	9606	2927	4192

Next page

8571
3300
9502
2929
2616
2796
6861
4558
8446
7734
2255
6351
51248 <-- C
8876
3534
9878
7044
5437
6268
5117
4605
41982
4559
3715
4255
9686
6125
9729
7622
3117
8300
9123
9819
7581
2477
8788
8377
5690
5598
3424
7853
9527
9784
6729

3061
9663
31925
4850 <-- C
2283
21595
5807
4265
6154
9522
4872
5841
5647
4601
9815
5964
2901
7090
2136
3483
5919
7276
5212
11610
2874
2036
5034
7079
7915
4819
2160
5977

APPENDIX B

Node 0 = .
Node 1 = 914
Node 2 = 827
Node 3 = 302
Node 4 = 1631
Node 5 = 785
Node 6 = 230
Node 7 = 11
Node 8 = 1567
Node 9 = 350
Node 10 = 1307
Node 11 = 1339
Node 12 = 929
Node 13 = 1216
Node 14 = 479
Node 15 = 703
Node 16 = 699
Node 17 = 90
Node 18 = 440 (ABOUT)
Node 19 = 1926
Node 20 = 1032
Node 21 = 1329
Node 22 = 1682
Node 23 = 1764
Node 24 = 1615 (ALL)
Node 25 = 1961
Node 26 = 1273
Node 27 = 1275
Node 28 = 38 (AN)
Node 29 = 923
Node 30 = 540
Node 31 = 1443
Node 32 = 1837
Node 33 = 1368
Node 34 = 1746
Node 35 = 1469
Node 36 = 505
Node 37 = 1480
Node 38 = 424
Node 39 = 678
Node 40 = 1139
Node 41 = 1763
Node 42 = 1959
Node 43 = 707
Node 44 = 242
Node 45 = 663
Node 46 = 1759
Node 47 = 332
Node 48 = 1455
Node 49 = 1685
Node 50 = 1716 (AS)
Node 51 = 1136
Node 52 = 1720
Node 53 = 1832
Node 54 = 751
Node 55 = 1681
Node 56 = 1106
Node 57 = 379
Node 58 = 1719
Node 59 = 381
Node 60 = 919
Node 61 = 1163
Node 62 = 219
Node 63 = 639
Node 64 = 1261
Node 65 = 40
Node 66 = 1144
Node 67 = 1941
Node 68 = 1872
Node 69 = 1569
Node 70 = 972
Node 71 = 1364
Node 72 = 1684
Node 73 = 931
Node 74 = 423
Node 75 = 1927
Node 76 = 1594
Node 77 = 182
Node 78 = 1401
Node 79 = 1868
Node 80 = 680
Node 81 = 538
Node 82 = 1940
Node 83 = 512 (ERASE)
Node 84 = 1289
Node 85 = 1621
Node 86 = 1970 (BEARD)
Node 87 = 1668
Node 88 = 1693
Node 89 = 352 (BLACK)
Node 90 = 940
Node 91 = 1208
Node 92 = 571
Node 93 = 1579
Node 94 = 821
Node 95 = 963 (BUTTONS)
Node 96 = 724
Node 97 = 762
Node 98 = 1187
Node 99 = 645
Node 100 = 86
Node 101 = 551
Node 102 = 329
Node 103 = 1018
Node 104 = 975
Node 105 = 608
Node 106 = 964
Node 107 = 1157 (CLINGS)
Node 108 = 1572
Node 109 = 366 (COAT)
Node 110 = 1377
Node 111 = 252
Node 112 = 1317 (THREE)
Node 113 = 764 (FOUR)
Node 114 = 545 (FIVE)
Node 115 = 1291 (SIX)
Node 116 = 735 (SEVEN)
Node 117 = 214 (EIGHT)
Node 118 = 1336 (NINE)
Node 119 = 1439 (ZERO)
Node 120 = 1154 (DRESSES)
Node 121 = 544 (START)
Node 122 = 362 (STOP)
Node 123 = 1085 (YES)
Node 124 = 1717 (NO)
Node 125 = 1325 (EVER)
Node 126 = 161 (GO)
Node 127 = 831 (HELP)
Node 128 = 806 (RUBOUT)
Node 129 = 918 (REPEAT)
Node 130 = 1416 (FROCK)
Node 131 = 1488 (GRANDFAT)
Node 132 = 1936 (ENTER)
Node 133 = 391 (HE)
Node 134 = 448 (M)
Node 135 = 1252 (L)
Node 136 = 1900 (H)
Node 137 = 20 (G)

Node 138 = 1618 (WELL)
Node 139 = 1630 (F)
Node 140 = 272 (E)
Node 141 = 522 (D)
Node 142 = 947 (C)
Node 143 = 1389 (B)
Node 144 = 1068 (A)
Node 145 = 590 (YEARS)
Node 146 = 476 (YET)
Node 147 = 1634 (YOU)
Node 148 = 267
Node 149 = 1140
Node 150 = 822
Node 151 = 629
Node 152 = 225
Node 153 = 891
Node 154 = 1954
Node 155 = 387
Node 156 = 1723 (HIMSELF)
Node 157 = 818
Node 158 = 1545
Node 159 = 641
Node 160 = 234
Node 161 = 1385
Node 162 = 977
Node 163 = 1834
Node 164 = 181
Node 165 = 878
Node 166 = 206
Node 167 = 1800
Node 168 = 508
Node 169 = 674
Node 170 = 1807
Node 171 = 220
Node 172 = 94 (IN)
Node 173 = 750
Node 174 = 1048
Node 175 = 80
Node 176 = 1028
Node 177 = 1315 (IS)
Node 178 = 1851
Node 179 = 1944
Node 180 = 1798 (KNOW)
Node 181 = 1094
Node 182 = 250
Node 183 = 537
Node 184 = 1420
Node 185 = 434
Node 186 = 1810
Node 187 = 1655
Node 188 = 1820
Node 189 = 739
Node 190 = 984
Node 191 = 6
Node 192 = 212
Node 193 = 1425
Node 194 = 1074 (LONG)
Node 195 = 1043
Node 196 = 99 (MISSING)
Node 197 = 882
Node 198 = 1263
Node 199 = 546
Node 200 = 1984
Node 201 = 626
Node 202 = 1012
Node 203 = 330
Node 204 = 1215
Node 205 = 701
Node 206 = 310
Node 207 = 876
Node 208 = 1238
Node 209 = 1127
Node 210 = 297 (MY)
Node 211 = 24
Node 212 = 1290
Node 213 = 304
Node 214 = 196
Node 215 = 1288
Node 216 = 202
Node 217 = 1654
Node 218 = 415
Node 219 = 1079
Node 220 = 1458
Node 221 = 1531
Node 222 = 163
Node 223 = 77
Node 224 = 147
Node 225 = 1737
Node 226 = 1055
Node 227 = 1160 (NEARLY)
Node 228 = 68
Node 229 = 1606
Node 230 = 375
Node 231 = 1046
Node 232 = 659
Node 233 =685 (NINETYTH)
Node 234 = 1923
Node 235 = 249
Node 236 = 165
Node 237 = 572
Node 238 = 274
Node 239 = 1228
Node 240 = 470 (OLD)
Node 241 = 188
Node 242 = 868
Node 243 = 673
Node 244 = 1843
Node 245 = 723
Node 246 = 1088
Node 247 = 922
Node 248 = 1653
Node 249 = 453
Node 250 = 1816
Node 251 = 1972 (SEVERAL)
Node 252 = 1964
Node 253 = 1710
Node 254 = 290
Node 255 = 1124
Node 256 = 130
Node 257 = 248
Node 258 = 1176
Node 259 = 908 (STILL)
Node 260 = 889
Node 261 = 1451
Node 262 = 1406
Node 263 = 1431
Node 264 = 1213
Node 265 = 1603
Node 266 = 254
Node 267 = 824
Node 268 = 927
Node 269 = 1244
Node 270 = 1547
Node 271 = 367
Node 272 = 519
Node 273 = 1200
Node 274 = 1265 (SWIFTLY)
Node 275 = 1324

Node 276 = 1369
Node 277 = 207
Node 278 = 1333
Node 279 = 1300
Node 280 = 849
Node 281 = 809
Node 282 = 1097
Node 283 = 1310
Node 284 = 357
Node 285 = 1901
Node 286 = 1114
Node 287 = 1515
Node 288 = 1489
Node 289 = 1298
Node 290 = 520
Node 291 = 1445 (THINKS)
Node 292 = 1904
Node 293 = 269
Node 294 = 1183
Node 295 = 1518
Node 296 = 168
Node 297 = 1550
Node 298 = 389
Node 299 = 1721
Node 300 = 1168
Node 301 = 1714
Node 302 = 1090
Node 303 = 758
Node 304 = 273
Node 305 = 775
Node 306 = 58
Node 307 = TO
Node 308 = 192
Node 309 = 1502
Node 310 = 616
Node 311 = 796
Node 312 = 861
Node 313 = 458
Node 314 = 446
Node 315 = 1734
Node 316 = 255
Node 317 = 351
Node 318 = 8 (USUALLY)
Node 319 = 1534
Node 320 = 1878
Node 321 = 1044 (WELL)
Node 322 = 1437
Node 323 = 268
Node 324 = 1117
Node 325 = 605
Node 326 = 1982
Node 327 = 559
Node 328 = 1715
Node 329 = 1686
Node 330 = 125 (WISH)
Node 331 = 1729
Node 332 = 1622
Node 333 = 317
Node 334 = 300
Node 335 = 1819
Node 336 = 1581
Node 337 = 477
Node 338 = 788
Node 339 = 377
Node 340 = 1690
Node 341 = 1598
Node 342 = 1424
Node 343 = 1853
Node 344 = 1527
Node 345 = 1784
Node 346 = 729
Node 347 = 1061
Node 348 = 1663
Node 349 = 850
Node 350 = 283
Node 351 = 1595
Node 352 = 265
Node 353 = 154 (YEARS)
Node 354 = 1522
Node 355 = 872
Node 356 = 1841
Node 357 = 1647
Node 358 = 601
Node 359 = 1815
Node 360 = 901
Node 361 = 136
Node 362 = 483
Node 363 = 1919
Node 364 = 1276
Node 365 = 1212
Node 366 = 1610
Node 367 = 874
Node 368 = 36
Node 369 = 1034
Node 370 = 1915
Node 371 = 819
Node 372 = 160
Node 373 = 637
Node 374 = 179
Node 375 = 232
Node 376 = 1987
Node 377 = 1509
Node 378 = 606
Node 379 = 580 (YET)
Node 380 = 1207
Node 381 = 747 (YOU)
Node 382 = 1825
Node 383 = 460
Node 384 = 1838
Node 385 = 313
Node 386 = 1295
Node 387 = 1757
Node 388 = 1589
Node 389 = 1367
Node 390 = 815
Node 391 = 896 (A)
Node 392 = 754
Node 393 = 246
Node 394 = 812
Node 395 = 1925
Node 396 = 141
Node 397 = 759
Node 398 = 1396
Node 399 = 320
Node 400 = 228
Node 401 = 1980
Node 402 = 307
Node 403 = 1962
Node 404 = 169
Node 405 = 568
Node 406 = 1651
Node 407 = 1678
Node 408 = 127
Node 409 = 399
Node 410 = 1855
Node 411 = 587
Node 412 = 237 (B)
Node 413 = 1883

Node 414 = 346
Node 415 = 1758
Node 416 = 926
Node 417 = 1639
Node 418 = 468
Node 419 = 1172
Node 420 = 451
Node 421 = 745
Node 422 = 592
Node 423 = 1504
Node 424 = 1679
Node 425 = 913
Node 426 = 1732
Node 427 = 1659
Node 428 = 1694 (C)
Node 429 = 731
Node 430 = 1741
Node 431 = 614
Node 432 = 382
Node 433 = 1771
Node 434 = 741
Node 435 = 781
Node 436 = 1627
Node 437 = 1680
Node 438 = 1796
Node 439 = 1563
Node 440 = 1906
Node 441 = 306
Node 442 = 832
Node 443 = 1946
Node 444 = 251
Node 445 = 397
Node 446 = 996
Node 447 = 640
Node 448 = 1342
Node 449 = 852 (D)
Node 450 = 671
Node 451 = 584
Node 452 = 682 (E)
Node 453 = 630
Node 454 = 1413
Node 455 = 1921
Node 456 = 1596
Node 457 = 44
Node 458 = 337
Node 459 = 577

Node 460 = 1671
Node 461 = 17
Node 462 = 1948
Node 463 = 1933
Node 464 = 1665
Node 465 = 1726
Node 466 = 591
Node 467 = 1448
Node 468 = 910
Node 469 = 1699
Node 470 = 1619
Node 471 = 1047
Node 472 = 611
Node 473 = 1899
Node 474 = 1236
Node 475 = 835
Node 476 = 1415
Node 477 = 1817
Node 478 = 1730
Node 479 = 1688
Node 480 = 85 (F)
Node 481 = 102
Node 482 = 456
Node 483 = 108
Node 484 = 471
Node 485 = 756
Node 486 = 1928
Node 487 = 773
Node 488 = 1348 (G)
Node 489 = 221
Node 490 = 1268
Node 491 = 1885
Node 492 = 1240
Node 493 = 1524
Node 494 = 1319
Node 495 = 1570
Node 496 = 1486
Node 497 = 757
Node 498 = 155
Node 499 = 1549
Node 500 = 339
Node 501 = 793
Node 502 = 95
Node 503 = 1264
Node 504 = 23
Node 505 = 21

Node 506 = 1649
Node 507 = 1958
Node 508 = 946 (H)
Node 509 = 1241
Node 510 = 1483
Node 511 = 617
Node 512 = 1093
Node 513 = 1988
Node 514 = 187
Node 515 = 82
Node 516 = 944
Node 517 = 1086
Node 518 = 1943
Node 519 = 1870
Node 520 = 1323
Node 521 = 1492
Node 522 = 209
Node 523 = 1479 (I)
Node 524 = 1354
Node 525 = 618
Node 526 = 43
Node 527 = 1949
Node 528 = 991
Node 529 = 496
Node 530 = 1064
Node 531 = 683
Node 532 = 1023
Node 533 = 820
Node 534 = 506
Node 535 = 1370
Node 536 = 846
Node 537 = 1976
Node 538 = 301
Node 539 = 928
Node 540 = 1272
Node 541 = 1739
Node 542 = 1223
Node 543 = 1142
Node 544 = 1062
Node 545 = 1703
Node 546 = 502
Node 547 = 547
Node 548 = 1057
Node 549 = 865
Node 550 = 473
Node 551 = 28

Node 552 = 909
Node 553 = 1371
Node 554 = 1808
Node 555 = 436
Node 556 = 1280
Node 557 = 491
Node 558 = 1811
Node 559 = 100
Node 560 = 416
Node 561 = 1890
Node 562 = 138
Node 563 = 218
Node 564 = 439 (J)
Node 565 = 1491
Node 566 = 178
Node 567 = 19
Node 568 = 985
Node 569 = 1593
Node 570 = 1087
Node 571 = 1041 (K)
Node 572 = 447
Node 573 = 1986
Node 574 = 450
Node 575 = 1312
Node 576 = 811
Node 577 = 830
Node 578 = 573
Node 579 = 1585
Node 580 = 331
Node 581 = 1745
Node 582 = 990
Node 583 = 1611
Node 584 = 63
Node 585 = 1005
Node 586 = 1470
Node 587 = 305
Node 588 = 1337
Node 589 = 524
Node 590 = 1935
Node 591 = 133
Node 592 = 466
Node 593 = 1704
Node 594 = 59
Node 595 = 1592
Node 596 = 1196
Node 597 = 170 (L)
Node 598 = 742
Node 599 = 378
Node 600 = 744
Node 601 = 1075
Node 602 = 425
Node 603 = 1922
Node 604 = 1038
Node 605 = 1014
Node 606 = 1453
Node 607 = 897
Node 608 = 1516
Node 609 = 235
Node 610 = 635
Node 611 = 368
Node 612 = 1002
Node 613 = 1917
Node 614 = 873
Node 615 = 706
Node 616 = 328
Node 617 = 1452 (M)
Node 618 = 1186
Node 619 = 749
Node 620 = 1000
Node 621 = 1357
Node 622 = 1378
Node 623 = 101
Node 624 = 171
Node 625 = 311
Node 626 = 597 (N)
Node 627 = 585
Node 628 = 566
Node 629 = 1963
Node 630 = 1661
Node 631 = 1983
Node 632 = 558
Node 633 = 1778
Node 634 = 1499 (O)
Node 635 = 413
Node 636 = 1303
Node 637 = 1514
Node 638 = 1767
Node 639 = 474
Node 640 = 1557
Node 641 = 1287
Node 642 = 658
Node 643 = 1030
Node 644 = 786
Node 645 = 150
Node 646 = 1360
Node 647 = 92
Node 648 = 25
Node 649 = 609
Node 650 = 1586
Node 651 = 622
Node 652 = 1363
Node 653 = 1180
Node 654 = 1493
Node 655 = 1421
Node 656 = 326
Node 657 = 259
Node 658 = 1076
Node 659 = 26
Node 660 = 2000 (P)
Node 661 = 841
Node 662 = 1482
Node 663 = 341
Node 664 = 1112
Node 665 = 649
Node 666 = 825
Node 667 =3387 (Q)
Node 668 = 675
Node 669 = 848
Node 670 = 83
Node 671 = 2914
Node 672 = 8277
Node 673 = 3062
Node 674 = 3631
Node 675 = 7845
Node 676 = 2380
Node 677 = 8011
Node 678 = 2350
Node 679 = 5307 (R)
Node 680 = 3339
Node 681 = 8929
Node 682 = 9216
Node 683 = 6479
Node 684 = 4703
Node 685 = 6999
Node 686 = 9000
Node 687 = 4480
Node 688 = 3926
Node 689 = 3329

Node 690 = 3682
Node 691 = 5764
Node 692 = 21615
Node 693 = 7961
Node 694 = 9273
Node 695 = 31275
Node 696 = 4038
Node 697 = 4923
Node 698 = 5490
Node 699 = 7443
Node 700 = 7837
Node 701 = 41368
Node 702 = 7746
Node 703 = 61469
Node 704 = 8505
Node 705 = 9480
Node 706 = 6424
Node 707 = 6678
Node 708 = 81139
Node 709 = 9763
Node 710 = 31959
Node 711 = 6707
Node 712 = 6242
Node 713 = 6663 (S)
Node 714 = 3759
Node 715 = 6332
Node 716 = 3455
Node 717 = 7685
Node 718 = 3716
Node 719 = 3136
Node 720 = 7720
Node 721 = 5832
Node 722 = 4751
Node 723 = 5681
Node 724 = 5106
Node 725 = 2379
Node 726 = 9719
Node 727 = 6381
Node 728 = 2919
Node 729 = 7163
Node 730 = 4219
Node 731 = 6389
Node 732 = 61261
Node 733 = 2040
Node 734 = 9144
Node 735 = 21941
Node 736 = 7872
Node 737 = 5569
Node 738 = 4972
Node 739 = 5364
Node 740 = 11684
Node 741 = 6931
Node 742 = 8423
Node 743 = 7927
Node 744 = 3594
Node 745 = 2182
Node 746 = 3401
Node 747 = 9868
Node 748 = 6820
Node 749 = 6538
Node 750 = 3940
Node 751 = 6512
Node 752 = 91289
Node 753 = 9621
Node 754 = 7970 (T)
Node 755 = 3668
Node 756 = 5693
Node 757 = 4352
Node 758 = 2940
Node 759 = 9208
Node 760 = 8571
Node 761 = 3579
Node 762 = 6821
Node 763 = 6963
Node 764 = 2724
Node 765 = 8762
Node 766 = 51187
Node 767 = 4645
Node 768 = 8600
Node 769 = 6551
Node 770 = 6329
Node 771 = 7018
Node 772 = 4975
Node 773 = 6080
Node 774 = 6964
Node 775 = 7157
Node 776 = 7572
Node 777 = 3606
Node 778 = 3377
Node 779 = 8252
Node 780 = 7317
Node 781 = 4764
Node 782 = 4545
Node 783 = 3291
Node 784 = 2735
Node 785 = 8214
Node 786 = 5336
Node 787 = 3439
Node 788 = 71154 (U)
Node 789 = 4544
Node 790 = 8362
Node 791 = 7717
Node 792 = 5325
Node 793 = 41617
Node 794 = 2831
Node 795 = 4806
Node 796 = 2918
Node 797 = 9416 (V)
Node 798 = 7488
Node 799 = 9936
Node 800 = 4391
Node 801 = 4448
Node 802 = 3252
Node 803 = 7900
Node 804 = 2020
Node 805 = 3618
Node 806 = 7630
Node 807 = 2762
Node 808 = 5222
Node 809 = 4947
Node 810 = 91389
Node 811 = 5068
Node 812 = 4590
Node 813 = 4760
Node 814 = 9634
Node 815 = 2697
Node 816 = 7140
Node 817 = 8202
Node 818 = 8629
Node 819 = 8225
Node 820 = 4891
Node 821 = 3954
Node 822 = 8387
Node 823 = 7723 (W)
Node 824 = 6818
Node 825 = 9545
Node 826 = 6410
Node 827 = 6234

Node 828 = 3385
Node 829 = 6929
Node 830 = 6977
Node 831 = 7834
Node 832 = 8190
Node 833 = 4878
Node 834 = 6206
Node 835 = 3800
Node 836 = 2508
Node 837 = 6740
Node 838 = 3807
Node 839 = 4220
Node 840 = 2094
Node 841 = 4750
Node 842 = 41048
Node 843 = 8080
Node 844 = 21028
Node 845 = 81315
Node 846 = 71572
Node 847 = 61851
Node 848 = 9944
Node 849 = 9798
Node 850 = 3094
Node 851 = 2500
Node 852 = 4537
Node 853 = 3420
Node 854 = 4434
Node 855 = 7810
Node 856 = 9655
Node 857 = 3820
Node 858 = 4739
Node 859 = 2984
Node 860 = 8006
Node 861 = 51273 (X)
Node 862 = 4214
Node 863 = 4212
Node 864 = 41425
Node 865 = 3074
Node 866 = 7043
Node 867 = 2099
Node 868 = 6882
Node 869 = 31263
Node 870 = 5546
Node 871 = 7984
Node 872 = 8663
Node 873 = 8626
Node 874 = 9012
Node 875 = 6330
Node 876 = 7215
Node 877 = 2626
Node 878 = 2701
Node 879 = 3310
Node 880 = 2876
Node 881 = 7238
Node 882 = 21127
Node 883 = 6297
Node 884 = 4024
Node 885 = 5290
Node 886 = 2304
Node 887 = 4196
Node 888 = 5288
Node 889 = 2202
Node 890 = 7654
Node 891 = 6415
Node 892 = 9079
Node 893 = 3458
Node 894 = 7531
Node 895 = 2163
Node 896 = 51074
Node 897 = 8077
Node 898 = 41497
Node 899 = 9737
Node 900 = 3055
Node 901 = 9160
Node 902 = 6068
Node 903 = 9606
Node 904 = 6375
Node 905 = 5046
Node 906 = 8659
Node 907 = 6685
Node 908 = 7923
Node 909 = 2249
Node 910 = 4165
Node 911 = 5792
Node 912 = 6274
Node 913 = 5807
Node 914 = 9228
Node 915 = 6470
Node 916 = 2188
Node 917 = 2868
Node 918 = 2673
Node 919 = 9843
Node 920 = 6723
Node 921 = 9088
Node 922 = 8922
Node 923 = 5653
Node 924 = 2546
Node 925 = 6453
Node 926 = 7816
Node 927 = 9972
Node 928 = 7964
Node 929 = 9710
Node 930 = 7290
Node 931 = 7124
Node 932 = 2130
Node 933 = 6248
Node 934 = 9851
Node 935 = 7176
Node 936 = 4908 (Y)
Node 937 = 2889
Node 938 = 21451
Node 939 = 3406
Node 940 = 8375
Node 941 = 3431
Node 942 = 9213
Node 943 = 7603
Node 944 = 2594
Node 945 = 11401
Node 946 = 4824
Node 947 = 2927
Node 948 = 91244
Node 949 = 81547
Node 950 = 8367
Node 951 = 6519
Node 952 = 7200
Node 953 = 7265
Node 954 = 9324
Node 955 = 71369
Node 956 = 7207
Node 957 = 9333
Node 958 = 3300
Node 959 = 6849
Node 960 = 2809
Node 961 = 3097
Node 962 = 5310
Node 963 = 4357
Node 964 = 21901
Node 965 = 3114

Node 966 = 11515
Node 967 = 71298
Node 968 = 4520
Node 969 = 5445 (Z)
Node 970 = 7904
Node 971 = 2274
Node 972 = 2689
Node 973 = 7183
Node 974 = 3518
Node 975 = 8168
Node 976 = 5550
Node 977 = 61721
Node 978 = 9168
Node 979 = 5714
Node 980 = 3090
Node 981 = 2758
Node 982 = 2273
Node 983 = 8775
Node 984 = 6058
Node 985 = 9122
Node 986 = 7936
Node 987 = 4192
Node 988 = 9502
Node 989 = 2929
Node 990 = 2616
Node 991 = 2796
Node 992 = 6861
Node 993 = 4558
Node 994 = 8446
Node 995 = 7734
Node 996 = 2255
Node 997 = 6351
Node 998 = 51248 (ONE)
Node 999 = 8876
Node 1000 = 3534
Node 1001 = 9878
Node 1002 = 7044
Node 1003 = 5437
Node 1004 = 6268
Node 1005 = 5117
Node 1006 = 4605
Node 1007 = 41982
Node 1008 = 4559
Node 1009 = 3715
Node 1010 = 4255
Node 1011 = 9686
Node 1012 = 6125
Node 1013 = 9729
Node 1014 = 7622
Node 1015 = 3117
Node 1016 = 8300
Node 1017 = 9123
Node 1018 = 9819
Node 1019 = 7581
Node 1020 = 2477
Node 1021 = 8788
Node 1022 = 8377
Node 1023 = 5690
Node 1024 = 5598
Node 1025 = 3424
Node 1026 = 7853
Node 1027 = 9527
Node 1028 = 9784
Node 1029 = 6729
Node 1030 = 3061
Node 1031 = 9663
Node 1032 = 31925
Node 1033 = 4850 (TWO)
Node 1034 = 2283
Node 1035 = 21595
Node 1036 = 4265
Node 1037 = 6154
Node 1038 = 9522
Node 1039 = 4872
Node 1040 = 5841
Node 1041 = 5647
Node 1042 = 4601
Node 1043 = 9815
Node 1044 = 5964
Node 1045 = 2901
Node 1046 = 7090
Node 1047 = 2136
Node 1048 = 3483
Node 1049 = 5919
Node 1050 = 7276
Node 1051 = 5212
Node 1052 = 11610
Node 1053 = 2874
Node 1054 = 2036
Node 1055 = 5034
Node 1056 = 7079
Node 1057 = 7915
Node 1058 = 4819
Node 1059 = 2160
Node 1060 = 5977

APPENDIX C

Planarity_breaking_arcs = from node 203 to 101
Planarity_breaking_arcs = from node 204 to 201
Planarity_breaking_arcs = from node 222 to 194
Planarity_breaking_arcs = from node 238 to 170
Planarity_breaking_arcs = from node 248 to 199
Planarity_breaking_arcs = from node 262 to 230
Planarity_breaking_arcs = from node 276 to 208
Planarity_breaking_arcs = from node 287 to 10
Planarity_breaking_arcs = from node 292 to 238
Planarity_breaking_arcs = from node 318 to 207
Planarity_breaking_arcs = from node 328 to 316
Planarity_breaking_arcs = from node 334 to 29
Planarity_breaking_arcs = from node 348 to 330
Planarity_breaking_arcs = from node 351 to 170
Planarity_breaking_arcs = from node 359 to 252
Planarity_breaking_arcs = from node 360 to 302
Planarity_breaking_arcs = from node 369 to 219
Planarity_breaking_arcs = from node 372 to 162
Planarity_breaking_arcs = from node 374 to 324
Planarity_breaking_arcs = from node 376 to 315
Planarity_breaking_arcs = from node 315 to 189
Planarity_breaking_arcs = from node 379 to 377
Planarity_breaking_arcs = from node 388 to 168
Planarity_breaking_arcs = from node 395 to 303
Planarity_breaking_arcs = from node 412 to 404
Planarity_breaking_arcs = from node 415 to 133
Planarity_breaking_arcs = from node 133 to 301
Planarity_breaking_arcs = from node 421 to 350
Planarity_breaking_arcs = from node 427 to 108
Planarity_breaking_arcs = from node 437 to 103
Planarity_breaking_arcs = from node 441 to 263
Planarity_breaking_arcs = from node 444 to 284
Planarity_breaking_arcs = from node 450 to 316
Planarity_breaking_arcs = from node 452 to 374
Planarity_breaking_arcs = from node 456 to 438
Planarity_breaking_arcs = from node 462 to 335
Planarity_breaking_arcs = from node 468 to 183
Planarity_breaking_arcs = from node 472 to 151
Planarity_breaking_arcs = from node 473 to 147
Planarity_breaking_arcs = from node 475 to 333
Planarity_breaking_arcs = from node 479 to 454
Planarity_breaking_arcs = from node 454 to 320
Planarity_breaking_arcs = from node 486 to 485
Planarity_breaking_arcs = from node 487 to 191
Planarity_breaking_arcs = from node 491 to 140

Planarity_breaking_arcs = from node 494 to 178
Planarity_breaking_arcs = from node 178 to 36
Planarity_breaking_arcs = from node 496 to 430
Planarity_breaking_arcs = from node 498 to 377
Planarity_breaking_arcs = from node 0 to 491
Planarity_breaking_arcs = from node 500 to 23
Planarity_breaking_arcs = from node 23 to 147
Planarity_breaking_arcs = from node 501 to 237
Planarity_breaking_arcs = from node 503 to 437
Planarity_breaking_arcs = from node 505 to 249
Planarity_breaking_arcs = from node 249 to 59
Planarity_breaking_arcs = from node 59 to 299
Planarity_breaking_arcs = from node 299 to 169
Planarity_breaking_arcs = from node 514 to 73
Planarity_breaking_arcs = from node 522 to 119
Planarity_breaking_arcs = from node 0 to 19
Planarity_breaking_arcs = from node 529 to 452
Planarity_breaking_arcs = from node 530 to 354
Planarity_breaking_arcs = from node 533 to 395
Planarity_breaking_arcs = from node 534 to 322
Planarity_breaking_arcs = from node 544 to 144
Planarity_breaking_arcs = from node 553 to 387
Planarity_breaking_arcs = from node 560 to 333
Planarity_breaking_arcs = from node 562 to 287
Planarity_breaking_arcs = from node 564 to 311
Planarity_breaking_arcs = from node 579 to 216
Planarity_breaking_arcs = from node 583 to 412
Planarity_breaking_arcs = from node 587 to 66
Planarity_breaking_arcs = from node 591 to 271
Planarity_breaking_arcs = from node 592 to 36
Planarity_breaking_arcs = from node 595 to 421
Planarity_breaking_arcs = from node 421 to 534
Planarity_breaking_arcs = from node 534 to 93
Planarity_breaking_arcs = from node 600 to 80
Planarity_breaking_arcs = from node 80 to 379
Planarity_breaking_arcs = from node 605 to 601
Planarity_breaking_arcs = from node 606 to 137
Planarity_breaking_arcs = from node 610 to 261
Planarity_breaking_arcs = from node 616 to 157
Planarity_breaking_arcs = from node 621 to 565
Planarity_breaking_arcs = from node 625 to 258
Planarity_breaking_arcs = from node 630 to 567
Planarity_breaking_arcs = from node 634 to 501
Planarity_breaking_arcs = from node 638 to 108
Planarity_breaking_arcs = from node 108 to 189
Planarity_breaking_arcs = from node 639 to 285
Planarity_breaking_arcs = from node 646 to 607
Planarity_breaking_arcs = from node 607 to 243

Planarity_breaking_arcs = from node 648 to 332
Planarity_breaking_arcs = from node 649 to 528
Planarity_breaking_arcs = from node 652 to 69
Planarity_breaking_arcs = from node 658 to 308
Planarity_breaking_arcs = from node 660 to 73
Planarity_breaking_arcs = from node 73 to 449
Planarity_breaking_arcs = from node 449 to 360
Planarity_breaking_arcs = from node 663 to 634
Planarity_breaking_arcs = from node 664 to 288
Planarity_breaking_arcs = from node 677 to 8
Planarity_breaking_arcs = from node 688 to 20
Planarity_breaking_arcs = from node 790 to 123
Planarity_breaking_arcs = from node 794 to 751
Planarity_breaking_arcs = from node 836 to 683
Planarity_breaking_arcs = from node 875 to 769
Planarity_breaking_arcs = from node 887 to 727
Planarity_breaking_arcs = from node 892 to 681
Planarity_breaking_arcs = from node 938 to 775
Planarity_breaking_arcs = from node 955 to 881
Planarity_breaking_arcs = from node 961 to 874
Planarity_breaking_arcs = from node 966 to 679
Planarity_breaking_arcs = from node 679 to 288
Planarity_breaking_arcs = from node 976 to 731
Planarity_breaking_arcs = from node 987 to 760
Planarity_breaking_arcs = from node 760 to 958
Planarity_breaking_arcs = from node 1035 to 913

APPENDIX D

ABOUT
ALL
AN
AS
BEARD
BLACK
BUTTONS
CLINGS
COAT
DRESSES
EVER
FROCK
GRANDFAT
HE
HIMSELF
IN
IS
KNOW
LONG
MISSING
MY
NEARLY
NINETYTH
OLD
SEVERAL
STILL
SWIFTLY
THINKS
TO
USUALLY
WELL
WISH
YEARS
YET
YOU
A
B
C
D
E
F
G
H

I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
ZERO
START
STOP
YES
NO
GO
HELP
ERASE
RUBOUT
REPEAT
ENTER

APPENDIX E

****
* The desired path we want is: *
****
s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
 - 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 133 - 134 - 135 - 136
 - 137 - 138 - 139 - 140 - 141 - 142 - 143 - 144 - 145 - 146 - 147 - *
****
****** The possible path we have searched ******
****
s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 201 - 205 - 206 - 207
 - 208 - 277 - 278 - 279 - 309 - 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168
 - 14 - 169 - 506 - 507 - 508 - 509 - 510 - 511 - 512 - 513 - 514 - *
stack = 1
likelihood = 197.062485

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 201 - 205 - 206 - 207
 - 208 - 277 - 278 - 279 - 309 - 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168
 - 14 - 169 - 170 - 239 - 240 - 241 - 242 - 243 - 244 - 245 - 246 - *
stack = 1
likelihood = 470.440613

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
 - 163 - 164 - 165 - 166 - 167 - 168 - 14 - 15 - 16 - 17 - 18 - 19 - 20
 - 689 - 690 - 691 - 692 - 693 - 694 - 695 - 696 - 697 - 698 - 699 - *
stack = 1
likelihood = 657.660461

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 46 - 47 - 48 - 49
 - 50 - 51 - 52 - 53 - 54 - 55 - 56 - 57 - 58 - 59 - 60 - 61 - 62
 - 63 - 64 - 65 - 66 - 67 - 68 - 69 - 70 - 71 - 72 - 73 - *
stack = 1
likelihood = 684.381226

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 46 - 47 - 48 - 49
 - 50 - 51 - 52 - 53 - 54 - 55 - 56 - 57 - 58 - 59 - 60 - 61 - 62
 - 63 - 64 - 65 - 66 - 67 - 68 - 69 - 653 - 654 - 655 - 656 - *
stack = 1
likelihood = 684.381226

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 201 - 205 - 206 - 207
 - 208 - 277 - 278 - 279 - 309 - 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168
 - 14 - 15 - 16 - 17 - 18 - 19 - 20 - 689 - 690 - 691 - 692 - *
stack = 1
likelihood = 698.163513

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 201 - 205 - 206 - 207
 - 208 - 277 - 278 - 279 - 309 - 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168
 - 14 - 15 - 16 - 17 - 18 - 19 - 525 - 526 - 527 - 528 - 529 - *
stack = 1
likelihood = 698.163513

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 201 - 205 - 206 - 207
- 208 - 277 - 278 - 279 - 309 - 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168
- 14 - 15 - 16 - 17 - 18 - 19 - 525 - 526 - 527 - 528 - 650 - *
stack = 1
likelihood = 698.163513

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
- 163 - 164 - 165 - 166 - 167 - 168 - 14 - 169 - 170 - 171 - 172 - 173 - 174
- 175 - 176 - 177 - 108 - 178 - 258 - 626 - 627 - 628 - 629 - 630 - *
stack = 3
likelihood = 1611.415649

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
- 163 - 164 - 165 - 166 - 167 - 168 - 14 - 169 - 170 - 171 - 172 - 173 - 174
- 175 - 176 - 177 - 108 - 428 - 429 - 430 - 431 - 432 - 433 - 434 - *
stack = 3
likelihood = 1616.514526

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
- 163 - 164 - 165 - 166 - 167 - 168 - 14 - 15 - 16 - 17 - 18 - 19 - 20
- 21 - 22 - 23 - 24 - 25 - 26 - 117 - 118 - 119 - 120 - 121 - *
stack = 3
likelihood = 1627.363525

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 201 - 205 - 206 - 207
- 208 - 277 - 278 - 279 - 309 - 12 - 13 - 14 - 169 - 170 - 171 - 172 - 173
- 174 - 175 - 176 - 177 - 108 - 428 - 429 - 430 - 431 - 432 - 433 - *
stack = 3
likelihood = 1941.516724

s - 37 - 38 - 39 - 40 - 41 - 42 - 43 - 44 - 45 - 201 - 205 - 206 - 207
- 208 - 277 - 278 - 279 - 309 - 12 - 13 - 14 - 169 - 170 - 171 - 172 - 173
- 174 - 175 - 176 - 177 - 108 - 178 - 258 - 626 - 627 - 628 - 629 - *
stack = 3
likelihood = 1996.617798

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
- 163 - 164 - 165 - 166 - 167 - 168 - 14 - 169 - 170 - 171 - 172 - 173 - 174
- 175 - 176 - 177 - 108 - 178 - 179 - 180 - 181 - 182 - 183 - 469 - *
stack = 3
likelihood = 2030.914551

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
- 163 - 164 - 165 - 166 - 167 - 168 - 14 - 15 - 16 - 17 - 18 - 19 - 20
- 21 - 22 - 23 - 24 - 25 - 26 - 27 - 28 - 29 - 30 - 31 - *
stack = 3
likelihood = 2100.762451

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
- 163 - 164 - 165 - 166 - 167 - 168 - 14 - 15 - 16 - 17 - 18 - 19 - 20
- 21 - 22 - 23 - 24 - 25 - 26 - 27 - 28 - 29 - 335 - 336 - *
stack = 3
likelihood = 2101.063477

s - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 162
- 163 - 164 - 165 - 166 - 167 - 168 - 14 - 15 - 16 - 17 - 18 - 19 - 20
- 21 - 22 - 23 - 24 - 25 - 26 - 27 - 28 - 29 - 335 - 463 - *
stack = 3
likelihood = 2101.063477

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168 - 14 - 15 - 16 - *
stack = 4
likelihood = 1815.964478

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168 - 14 - 169 - 170 - *
stack = 4
likelihood = 1816.265503

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168 - 14 - 169 - 506 - *
stack = 4
likelihood = 1816.265503

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 133 - 134 - 135 - 136
- 137 - 138 - 139 - 140 - 141 - 142 - 143 - 144 - 145 - 146 - 147 - *
stack = 5
likelihood = 2005.385254

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 133 - 134 - 135 - 136
- 137 - 138 - 139 - 140 - 141 - 142 - 143 - 144 - 545 - 546 - 547 - *
stack = 5
likelihood = 2005.385254

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 169 - 506 - 507 - 508 - 509 - 510 - 511 - 512 - *
stack = 5
likelihood = 2056.364502

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 169 - 170 - 239 - 240 - 241 - 242 - 243 - 244 - *
stack = 5
likelihood = 2448.242676

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 169 - 170 - 239 - 240 - 241 - 242 - 243 - 647 - *
stack = 5
likelihood = 2448.242676

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 93 - 94
- 95 - 96 - 97 - 98 - 99 - 100 - 101 - 102 - 103 - 104 - 105 - *
stack = 5
likelihood = 2595.387207

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 93 - 94
- 95 - 96 - 97 - 98 - 99 - 100 - 101 - 102 - 103 - 438 - 439 - *
stack = 5
likelihood = 2595.688232

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 93 - 94
- 95 - 96 - 97 - 98 - 99 - 100 - 101 - 102 - 103 - 438 - 457 - *
stack = 5
likelihood = 2595.688232

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 15 - 16 - 17 - 18 - 19 - 525 - 526 - 527 - *
stack = 5
likelihood = 2768.864502

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 15 - 16 - 17 - 18 - 19 - 20 - 21 - 22 - *
stack = 5
likelihood = 2769.165527

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 15 - 16 - 17 - 18 - 19 - 20 - 689 - 690 - *
stack = 5
likelihood = 2769.165527

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 169 - 170 - 171 - 172 - 173 - 174 - 175 - 176 - *
stack = 5
likelihood = 2856.441650

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 169 - 170 - 352 - 353 - 354 - 355 - 356 - 357 - *
stack = 5
likelihood = 2856.742676

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 13 - 14 - 169 - 170 - 352 - 353 - 354 - 531 - 532 - 533 - *
stack = 5
likelihood = 2856.742676

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 128 - 129 - 130 - 131 - 132 - 308 - 92 - 279 - 309
- 12 - 162 - 163 - 164 - 165 - 166 - 167 - 168 - 389 - 390 - 391 - *
stack = 5
likelihood = 3199.663574

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 84 - 85 - 86 - 87 - 88 - 89 - 90 - 91 - 92
- 279 - 309 - 12 - 13 - 14 - 169 - 170 - 352 - 353 - 354 - 355 - *
stack = 5
likelihood = 3469.940430

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 84 - 85 - 86 - 87 - 88 - 89 - 90 - 91 - 92
- 279 - 309 - 12 - 13 - 14 - 169 - 170 - 352 - 353 - 354 - 531 - *
stack = 5
likelihood = 3469.940430

s - 112 - 113 - 114 - 115 - 116 - 117 - 118 - 119 - 120 - 121 - 122 - 123 - 124
- 125 - 126 - 127 - 83 - 84 - 85 - 86 - 87 - 88 - 89 - 90 - 91 - 92
- 93 - 94 - 95 - 96 - 97 - 98 - 99 - 100 - 101 - 102 - 103 - *
stack = 5
likelihood = 3505.184326