FTL REPORT R82-7

ALGORITHMIC APPROACHES TO CIRCUIT ENUMERATION PROBLEMS AND APPLICATIONS

Boon Chai Lee

June 1982

MIT

DEPARTMENT OF AERONAUTICS & ASTRONAUTICS

FLIGHT TRANSPORTATION LABORATORY
Cambridge, Mass. 02139

FTL REPORT R82-7

ALGORITHMIC APPROACHES TO CIRCUIT ENUMERATION PROBLEMS AND APPLICATIONS

Boon Chai Lee

June 1982

ALGORITHMIC APPROACHES TO CIRCUIT ENUMERATION

PROBLEMS AND APPLICATIONS

by

BOON CHAI LEE

B.S., University of Michigan
(1978)

SUBMITTED TO THE DEPARTMENT OF
AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS OF THE
DEGREE OF

MASTER OF SCIENCE IN
AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1982

Signature of Author _____
Department of Aeronautics and Astronatics
May 7, 1982

Certified by _____
Robert W. Simpson
Thesis Supervisor

Accepted by _____
Harold Y. Wachman
Chairman, Departmental Graduate Committee

ALGORITHMIC APPROACHES TO CIRCUIT ENUMERATION PROBLEMS AND APPLICATIONS

by

BOON CHAI LEE

Submitted to the Department of Aeronautics and Astronautics
on May 7, 1982 in partial fulfillment of the
requirements for the Degree of Master of Science in
Aeronautics and Astronautics

ABSTRACT

A review of methods of enumerating elementary cycles and circuits is presented. For the directed planar graph, a geometric view of circuit generation is introduced making use of the properties of dual graphs. Given the set of elementary cycles or circuits, a particular algorithm is recommended to generate all simple circuits. A simple example accompanies each of the methods discussed. Some methods of reducing the size of the graph but maintaining all circuits are introduced. Worst-case bounds on computational time and space are also given.

The problem of enumerating elementary circuits whose cost is less than a certain fixed cost is solved by modifying an existing algorithm. The cost of a circuit is the sum of the cost of the arcs forming the circuit where arc costs are not restricted to be positive. Applications of circuits with particular properties are suggested.

Thesis Supervisor: Robert W. Simpson

Title: Professor of Aeronautics and Astronautics

ACKNOWLEDEMENTS

Professor Simpson had been most helpful with his comments and directions throughout the duration of this thesis. He is also always willing to share his invaluable time for discussions. He helped me present my thesis clearer and better and showed interest in this work throughout. I owe much to him for his gentleness and generousity.

My stay here in the United States was possible from the support of my family. They have been very generous and loving and I missed them alot. I am very grateful that they have provided me with the chance not only to further my education but also to introduce me to such a variety of people and habits that continues to entreat me even now.

Next, I would like to thank Yutaka Takase for the excellent diagrams he drew for me, and to Wu, Yiren and Hirofumi Matsuo for countless hours of interesting discussions.

Joanne Clark also helped with proof reading this thesis and showed much patience.

This thesis was completed with the help also of Debbi who spent several beautiful Spring days typing.

TABLE OF CONTENTS

Section 1

Introduction

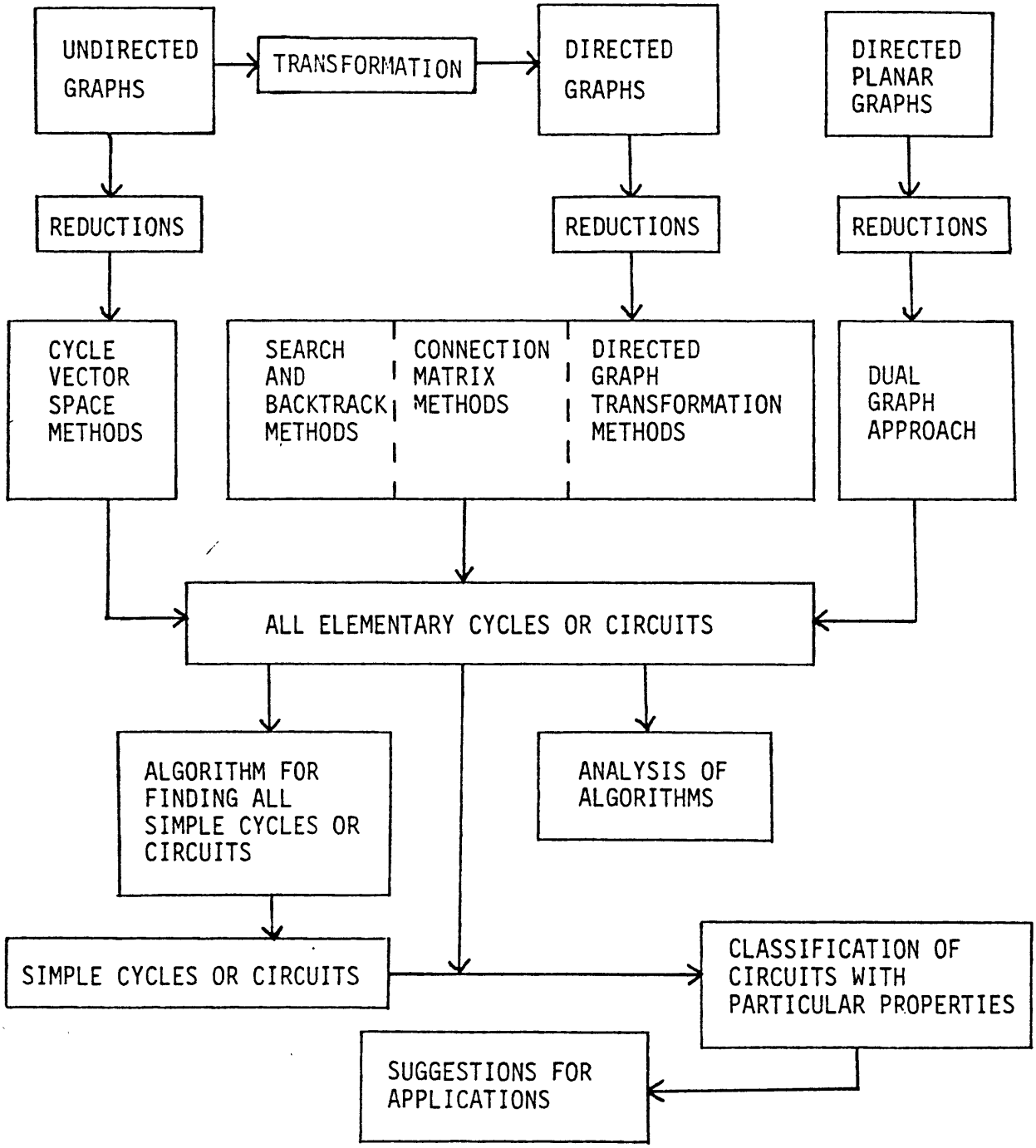
The circuit enumeration problem is of theoretical as well as practical interest. In the past years, we have seen a wide range of researchers from many disciplines working on this topic. These areas include, Mathematics [3], Computer Science [1,18,34,32], Medicine [25], Transportation [9,21], and Engineering [19,36]. The reason is that many problems can be represented as graphs. Furthermore, many problems have a cyclic structure where the problem is to identify some or all of the circuits or cycles in the graph.

In particular, this thesis addresses the problem of finding all elementary circuits and cycles in a graph and suggests some related applications. The areas we will be covering are represented in fig. 1.1.

In Section 2, we create four classes of methods of generating elementary circuits. Every algorithm for generating elementary circuits known thus far, belongs to one of the four methods; namely, the Cycle Vector Space Methods; the Search and Backtrack Methods; the Connection Matrix Methods, and the Directed Graph Transformation Methods. In addition, we provide an algorithm for generating all simple cycles or circuits given the set of all elementary cycles or circuits. If the directed graph is planar, we introduce a method for enumerating all the elementary circuits using the dual graphs.

Section 3 deals with an analysis of all the algorithms on circuit

fig. 1.1 Structured approach to generating all cycles or circuits and applications.



enumeration. Before comparing algorithms, the graphs are edited and reduced. Several ways of reducing and editing graphs are given. Thereafter, the running time and storage requirement of each algorithm is given followed by a short discussion, and recommendations for the best algorithms.

Since complete elementary circuit enumeration problems are known to be difficult and intractable, we have, in section 4, identified a list of circuits with particular properties which, hopefully, greatly reduces the number of circuits to be enumerated, followed by some suggested applications that fit into our problem classification. An example would be finding all elementary circuits that do not exceed a certain fixed cost. (The cost of a circuit is the sum of the cost of all arcs in that circuit).

In the final section, we summarize what we have done, and point to some interesting areas of research.

For completion, we have included Appendices to find "strong components" of a graph and a treatment of the problems of finding Eulerian and Hamiltonian circuits.

1.1 Definitions

The terminologies used in graph theory have hitherto remained at the discretion of the writer. Some writers choose an arc over an edge, a link over a line, or a path over a chain, etc. Furthermore, there have been additions to the glossary of terms used; for instance with flowers, came blossoms, with spanning tree, came forest. Then there are branches

and fronds and twigs, and so forth. As a result, there is a need in this subsection to define certain terms that will be used throughout this thesis. Many ambiguities would be clarified if the reader would take a minute to browse through this subsection.

A graph G(V,E) is defined as a finite set of vertices V and edges E which connects pairs of vertices. The number of vertices $|V|=n$ and edges $|E|=e$ for a graph G(V,E). A directed graph has arcs which are edges with directions associated with them. A directed graph is more commonly represented by $G(V,\Gamma)$ where V is the set of vertices and P is the vertex operator, where for $i\varepsilon V$, $j\varepsilon V$, $j\varepsilon\Gamma(i)$ if an arc ij exists. Two vertices are said to be adjacent if they are connected by a common edge. Two edges with a common vertex are said to be adjacent.

A path is a directed or undirected sequence of edges or arcs where the final vertex of one is the initial vertex of the next edge or arc. A simple path is a path which does not use the same edge or arc more than once. A simple cycle is a simple path where the initial and final vertex coincide, and the edges contained in the path are assumed to be undirected. A circuit is a directed version of a cycle. An elementary path is a path which does not use the same vertex more than once. An elementary cycle is an elementary path where the initial and final vertex coincide. An elementary circuit is a directed version of an elementary cycle. All elementary paths and cycles or circuits also must be simple. Two elementary cycles or circuits are distinct if one is not a cyclic permutation of the other. We refer to cycles or circuits that are neither elementary nor simple as infinite cycles and circuits respectively.

In this work, we will not be concerned with infinite cycles or circuits.

The length or cardinality of a path, cycle or circuit is the number of edges or arcs appearing in it. A path of length k is called a k-path. The same applies to cycles or circuits.

For a directed graph, the indegree and outdegree of a vertex is the number of arcs terminating and originating at that vertex. The degree of a vertex in an undirected graph is the number of arcs incident to it.

A vertex i is connected to j if there exists a path from i to j. A connected undirected graph is one for which a path exists between every pair of vertices. Similarly, a directed graph is connected if its associated undirected graph is connected. Note there may not be a directed path connecting all vertices in a directed graph. In our work, we shall only be concerned with connected graphs.

A subgraph $G(X,A)$ of a graph $G(V,E)$ is a graph such that $X \subset V$ and $A \subset E$. A tree of an undirected graph is a connected subgraph which has no cycles. A spanning tree of a graph is a tree of the graph that contains all the vertices.

Any other definitions that are necessary will be introduced subsequently.

Section 2

Review of Circuit and Cycle Enumeration Methods

This section reviews the different methods for generation of the elementary cycles/circuits in a graph. All algorithms known thus far for enumerating elementary cycles/circuits can be classified into one of the following methods:

2.1 Cycle Vector Space Methods.

2.2 Search and Backtrack Methods.

2.3 Connection Matrix Methods.

2.4 Directed Graph Transformation Methods.

The purpose here is to explore the underlying idea behind each of these methods by referring to explicit algorithms.

Thereafter, we examine procedures for generation of all the simple circuits in a graph given the set of elementary circuits. An easy procedure to do this is recommended.

Due to close associations of the Travelling Salesman and the Chinese Postman Problem to the Hamiltonian and Eulerian circuit generation, we have included a rather complete discussion on the enumeration of Hamiltonian and Eulecian circuits in Appendix B. We have not included the discussion in this section in order to reduce the amount of redundancy since the enumeration of Hamiltonian and Eulecian circuits are special cases of the cycle/circuit enumeration methods we will be dealing with in this section.

Finally, this section concludes by introducing a method of generating all elementary circuits in a directed planar graph using vertex aggregation of the associated dual graph.

2.1 Cycle Vector Space Methods

These methods apply to an undirected graph and finds all elementary and/or simple cycles. Given an undirected graph G(V,E), a spanning tree Tg is first constructed having n-1 edges. Tg is not unique. The addition to Tg of any edge in G (but not in Tg) will form a unique elementary cycle. Every cycle formed in this way contains at least one edge not found in another. The set of cycles formed by adding all edges in G (but not in Tg) provides a basis for the vector space of all the simple cycles in the graph. Since there are e edges in G and n-1 edges in Tg, the number of such cycles that will be formed is e-n+1. This is the cyclomatic number of G, $\nu(G)$. The $\nu(G)$ cycles formed in this way are known as a fundamental set of cycles

The derivation of the fundamental set of cycles is not difficult and therefore, the reader is referred to Gotlieb and Corneil [15], Welch [36] or Paton [25] for an algorithm for finding the fundamental set of cycles. The algorithm of Gotlieb and Corneil [15] is slower than that of Welch [36] but requires less storage for graphs with a large number of vertices. The algorithm of Paton [25] on the other hand is compatible to Gotlieb and Corneil [15] in terms of storage and to Welch [36] in terms of speed. The author recommends the algorithm of Paton for finding the fundamental set of cycles.

Since the fundamental set of cycles is a basis of the vector space for cycles, then any cycle in G not in $\underline{\Phi}$, can be formed by a linear combination of cycles in $\underline{\Phi}$ by the following convention.

Let every fundamental cycle $\underline{\Phi}_i$, i=1,2,...ν(G), be represented by an e dimensional vecter where the j^{th} element is 1 if the j^{th} edge is part of the cycle, and zero otherwise. The ring sum operation can be expressed for vectors A and B as $A + B = \{x | x \in A \cup B, \notin A \cap B\}$. If the ring sum, $\oplus$, is used for mod 2 addition, any cycles in G not in $\underline{\Phi}$, can be expressed as a ring sum operation of fundamental cycle. The ring sum of 2 cycles is a cycle or an edge disjoint union of cycles.[1] The edge disjoint union of cycles means the union of cycles having no common edges. To generate all the cycles in G, we need to consider all $2^{\nu(G)}-\nu(G)-1$ combinations of fundamental cycles. However, some of the combinations will be disjoint cycles, but, if a given combination is disjoint one cannot disregard other combinations containing it since the mod 2 addition of it and another combination might produce a single cycle. Thus, one can simply enumerate all possible cycles in the graph using this property of the fundamental set of cycles found by selecting a spanning tree.

Gibbs [14] presented a corrected version of Welch's algorithm to generate all the elementary cycles in the graph. The algorithm is presented here as ALGORITHM 1 for completion.

We note that the set R generally remains smaller than Q which contains all the linear combinations of fundamental cycles at the end. This is more suitable for programming.

To illustrate the above method, an example is provided in Example 1.

ALGORITHM 1. GIBB's algorithm for generating all elementary cycles from the fundamental set of cycles.

Given a set of fundamental cycles $\underline{\Phi} = \{ \Phi_1, \Phi_2, \ldots \Phi_{\upsilon(G)} \}$

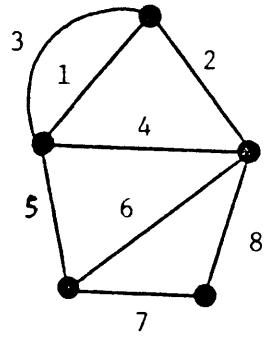
1. Set $S = \{\Phi_1\}$, $Q = \{\Phi_1\}$, $R = \emptyset$, $R^* = \emptyset$, $i = 2$.
2. For all T in Q,
   If $T \cap \Phi_i \neq \emptyset$ place $T \oplus \Phi_i$ into R,
   If $T \cap \Phi_i = \emptyset$ place $T \oplus \Phi_i$ into R*.
3. For all U and V in R, if $U \subset V$ set $R = R - \{ V \}$ and $R^* = R^* \cup \{ V \}$.
4. Set $S = S \cup R \cup \{ \Phi_i \}$.
5. Set $Q = Q \cup R \cup R^* \cup \{ \Phi_i \}$. Reset $R = \emptyset$; Reset $R^* = \emptyset$.
6. Set $i = i + 1$. If $i \leq \upsilon(G)$, go to 2. If $i > \upsilon(G)$, STOP; S consists of all the elemenatary cycles in the graph.

Note: T is any element of Q, i.e., it could be a cycle or an edge disjoint union of cycles.

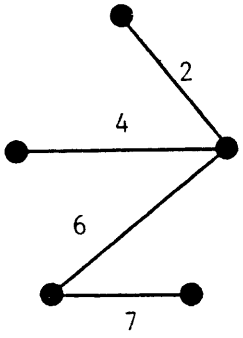
End of ALGORITHM 1

Example 1. Generating all elementary cycles using Cycle Vector Space Methods.

Given: G,

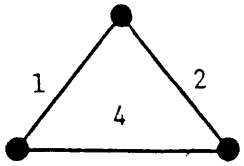


T_g

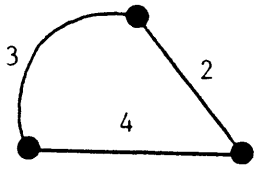


We obtain the fundamental set of cycles :-

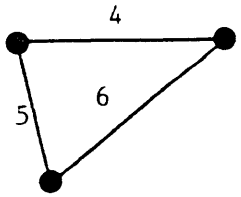
Φ_1 : Adding edge 1



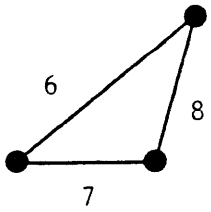
Φ_2 : Adding edge 3



Φ_3 : Adding edge 5



Φ_4 : Adding edge 8



$\underline{\Phi}$ =

	1	2	3	4	5	6	7	8
Φ_1	1	1	0	1	0	0	0	0
Φ_2	0	1	1	1	0	0	0	0
Φ_3	0	0	0	1	1	1	0	0
Φ_4	0	0	0	0	0	1	1	1

Obtaining all elementary cycles from the fundamental set using Gibb's algorithm:

Iteraion	Results	Action
1	$S = \Phi_1, \quad Q = \Phi_1, \quad R = \emptyset, \quad R^* = \emptyset$	(STEP 1)
2	$\Phi_1 \cap \Phi_2 \neq \emptyset \quad \Rightarrow \quad R = \Phi_1 \oplus \Phi_2$	(STEP 2)
3	$S = \Phi_1, \Phi_1 \oplus \Phi_2, \Phi_3$	(STEP 4)
4	$Q = \Phi_1, \Phi_1 \oplus \Phi_2, \Phi_2, R = \emptyset, \quad R^* = \emptyset$	(STEP 5)
5	$i = 3, \quad 3 \leq \upsilon(G) = 4$	(STEP 6)
6	$\Phi_1 \cap \Phi_3 \neq \emptyset$ $(\Phi_1 \oplus \Phi_2) \cap \Phi_3 = \emptyset$ $\Rightarrow$ $R = \Phi_1 \oplus \Phi_3, \Phi_2 \oplus \Phi_3$ $\Phi_2 \cap \Phi_3 \neq \emptyset$ $\quad R^* = \Phi_1 \oplus \Phi_2 \oplus \Phi_3$	(STEP 2)
7	$S = \Phi_1, \Phi_1 \oplus \Phi_2, \Phi_1 \oplus \Phi_3, \Phi_2 \oplus \Phi_3, \Phi_3$	(STEP 4)
8	$Q = \Phi_1, \Phi_1 \oplus \Phi_2, \Phi_2, \Phi_1 \oplus \Phi_3, \Phi_1 \oplus \Phi_2 \oplus \Phi_3,$ $\Phi_2 \oplus \Phi_3, \Phi_3.$ $R = \emptyset, \quad R^* = \emptyset.$	(STEP 5)
9	$i = 4, \quad 4 \leq \upsilon(G) = 4$	(STEP 6)
10	$\Phi_1 \cap \Phi_4 \neq \emptyset, \quad \Phi_2 \cap \Phi_4 = \emptyset,$ $(\Phi_1 \oplus \Phi_2) \cap \Phi_4 = \emptyset, \quad (\Phi_1 \oplus \Phi_2 \oplus \Phi_3) \cap \Phi_4 \neq \emptyset,$ $(\Phi_1 \oplus \Phi_3) \cap \Phi_4 \neq \emptyset, \quad (\Phi_2 \oplus \Phi_3) \cap \Phi_4 \neq \emptyset,$ $\Phi_3 \cap \Phi_4 \neq \emptyset.$ $\Rightarrow R = \{ \Phi_1 \oplus \Phi_3 \oplus \Phi_4, \Phi_1 \oplus \Phi_2 \oplus \Phi_3 \oplus \Phi_4,$ $\Phi_2 \oplus \Phi_3 \oplus \Phi_4, \Phi_3 \oplus \Phi_4 \}$ $R^* = \{ \Phi_1 \oplus \Phi_4, \Phi_1 \oplus \Phi_2 \oplus \Phi_4, \Phi_2 \oplus \Phi_4 \}$	(STEP 2)
11	In this case, we enter step 3, $\Phi_3 \oplus \Phi_4 \subset \Phi_1 \oplus \Phi_2 \oplus \Phi_3 \oplus \Phi_4$ since (0 0 0 1 1 0 1 1) (1 0 1 1 1 0 1 1) i.e. (1 0 1 0 0 0 0 0) U (0 0 0 1 1 0 1 1), therefore, $R = \{ \Phi_1 \oplus \Phi_3 \oplus \Phi_4, \Phi_2 \oplus \Phi_3 \oplus \Phi_4, \Phi_3 \oplus \Phi_4 \}$ $R^* = \{ \Phi_1 \oplus \Phi_4, \Phi_1 \oplus \Phi_2 \oplus \Phi_4, \Phi_2 \oplus \Phi_4,$ $\Phi_1 \oplus \Phi_2 \oplus \Phi_3 \oplus \Phi_4 \}$	(STEP 3)

Iteration	Results	Action
12	$S = \Phi_1,\ \Phi_1 \oplus \Phi_2,\ \Phi_2,\ \Phi_1 \oplus \Phi_3,\ \Phi_2 \oplus \Phi_3,$ $\Phi_3,\ \Phi_1 \oplus \Phi_3 \oplus \Phi_4,\ \Phi_2 \oplus \Phi_3 \oplus \Phi_4,$ $\Phi_3 \oplus \Phi_4,\ \Phi_4.$	(STEP 4)
13	$Q = \Phi_1,\ \Phi_1 \oplus \Phi_2,\ \Phi_2,\ \Phi_1 \oplus \Phi_3,\ \Phi_1 \oplus \Phi_2 \oplus \Phi_3,$ $\Phi_2 \oplus \Phi_3,\ \Phi_3,\ \Phi_1 \oplus \Phi_3 \oplus \Phi_4,\ \Phi_2 \oplus \Phi_3 \oplus \Phi_4,$ $\Phi_3 \oplus \Phi_4,\ \Phi_1 \oplus \Phi_4,\ \Phi_1 \oplus \Phi_2 \oplus \Phi_4,\ \Phi_2 \oplus \Phi_4,$ $\Phi_1 \oplus \Phi_2 \oplus \Phi_3 \oplus \Phi_4,\ \Phi_4.$ $R = \emptyset,\ R^* = \emptyset$	(STEP 5)
14	$i = 5 \quad 5 \geq \upsilon(G) = 4.$ STOP.	(STEP 6)

S in iteration 12 contains all the elementary cycles in the graph, and Q in 13 all the linear combinations of the fundamental cycles.

********** End of Example 1 **********

The edges are labeled from 1 to e=8. Tg is one of the spanning trees of the graph G.

Example 1 illustrates the enumeration of all elementary cycles from the fundamental set of cycles. However, the set of fundamental cycle forms a basis for all the simple cycles of the graph as well. These simple but non-elementary cycles can be found in set Q. One way of obtaining these will be discussed in section 2.5.

Thus far, we have restricted our discussion to an undirected graph only. For a directed graph, there are no equivalent fundamental sets of circuits. Under the ring sum operation, + , cycles and edge disjoint union of cycles in the undirected graph form a group. Every element of this group can be expressed as a ring sum of some of the fundamental cycles with respect to a spanning tree. However, there exists no binary operation under which all circuits and edge disjoing unions of circuits form a group, let alone a vecter space. (Narsingh Deo [10]). Cycle Vector Space Methods are not applicable to a directed graph.

We note from the above discussion that the fundamental set of cycles is central to cycle enumeration. In the discussion following we point out two interesting relationships between the fundamental cycle matrix and the cutset and incidence matrix. These relationships not only offer better insight into the properties of the fundamental cycle set but also application to other seemingly unrelated problems. A few definitions are necessary before we proceed. The fundamental cutset[2] matrix is defined as an (n-1) x e matrix $K = [k_{ij}]$ where $k_{ij} = 1$ if edge j is part of cutset k_i and 0 otherwise. Similarly, and incidence matrix is defined

as an n x e matrix B = [bij] where bij = 1 if vertex i is incident to edge j and zero otherwise. We state the two relationships under theorems one and two.

Theorem 1. The incidence matrix B and the transpose of the fundamental cycle matrix $\underline{\Phi}^T$ are orthogonal, i.e., $B \cdot \underline{\Phi}^T = 0$.

Theorem 2. The fundamental cycle matrix $\underline{\Phi}$ and the cutset matrix K^T are also orthogonal, i.e., $\underline{\Phi} \cdot K^T = 0$.

The two theorems can be shown easily by observing that:-

1. Each vertex in a cycle is incident with an even number of edges in the cycle.
2. Each cycle cut by a cutset has an even number of edges in common with the cutset.

Observe that all operations are done in mod. 2. Moveover, the theorems are valid for any cycle and cutset matrices defined so long as matrix multiplication rule is not violated. The fundamental cycle set have been used to solve electrical circuit problems (see Christofides [7]). In addition, from the max-flow-min-cut theorem for the maximum flow problem, one would search for the min-cut by observing the relationship between the fundamental cycles and the fundamental cutsets. Note that this orthogonality relationship extends beyond the fundamental sets to include all the cycles and cutsets in any given graph.

One of the drawbacks of the Cycle Vector Space Methods stem from the fact that in most cases, the ratio of the number of nondegenerate or valid cycles to the number of vectors goes asymptotically to zero

as the number of vertices in the graph increases. The number of vertices in the graph increases. The number of vectors is $2^{\upsilon(G)} - \upsilon(G) - 1$ (excluding the basic cycles and the null elements). The ring sum, $\oplus$, taken on combinations of fundamental cycles thus produces many degenerate cycles or edge disjoint unions of cycles. In fact, it is shown in Mateti and Deo [23] that there are only four graphs having all $2^{\nu(G)} - 1$ cycles, that is, every combination of fundamental cycles produces a distinct cycle.

Algorithms which make use of this method were given by Welch [36], Gibbs [14], Mateti and Deo [23], Maxwell and Reed [24], and Hsu and Honkanen [17]. The discussion above attempts to capture the essence of the Cycle Vector Space Methods.

2.2. Search and Backtrack Methods

The search and backtrack method applies to a directed graph only. One such algorithm is presented here to introduce the main idea behind the Search and Backtrack Methods.

The vertices of a directed graph are numbered 1 to $|V| = n$. The algorithm generates all elementary paths $P = \{P(1),P(2),P(3),\ldots P(k)\}$ where $P(i)$ is the i^{th} vertex in the k-path P and $P(1) < P(i)$ for all $2 \leq i \leq k$ by starting from an arbitrary vertex $P(1)$, choosing an arc to extend to another vertex $P(2) > P(1)$, and continuing in this way. If the path cannot be extended any further, the procedure backs up one vertex and chooses to extend to a different vertex. If $P(1)$ is

adjacent to P(k), the algorithm lists an elementary circuit (P(1),P(2),...,P(k),P(1)). This algorithm enumerates each elementary circuit exactly once, since each circuit contains a unique "initial vertex," P(1), and thus corresponds to a unique elementary path starting from that vertex.

Given a directed graph, we first reduce the size of the graph by eliminating vertices which cannot belong to any circuits. The process is to remove any vertices on which no arcs terminate and all arcs originating from these vertices. Similarly, all vertices in which no arc originates, and any arcs terminating on these vertices are also removed. This step is repeated until no such vertices remain. Then, k parallel arcs are reduced to a single arc, but k circuits are later listed if that particular arc forms part of any circuit. More discussions on graph reduction will appear in Section 3.1. The reduced directed subgraph $G(V,\Gamma)$ is defined as a set of vertices $V = (1,2,...n)$ and an arc operator $\Gamma(\cdot)$ which operates on all elements of V; $j \in \Gamma(i)$ if there exists an arc from i to j. Graph $G(V,\Gamma)$ is an $n \times n$ array G(i,j) (see Example 2).

The algorithm assigns integer values, 1,2,...,n to each of the vertices in G. It utilizes two principal arrays in addition to G(i,j). The first, P, is an $n \times 1$ array, containing all the vertices in an elementary path. The second is an $n \times n$ array, H, which is initially zeroed. H contains the list of vertices "closed", to each vertex. Vertex j is "closed" to i whenever an arc from i to j has been previously considered. The algorithm

basically involves elementary path building in array, P. We can now explain the algorithmic process.

Search

Starting from an "initial" vertex 1, a path is extended from its end, one arc at a time such that:-

a. The extension vertex cannot be already in P.

b. The extension vertex value must be larger than the initial vertex.

c. The extension vertex is not "closed" to the last vertex in P. H contains the list of vertices closed to each vertex. (Vertex closure will be discussed further under *Backtrack*).

(a) assures that an elementary path is being considered. (b) assures that each circuit will be considered only once. (c) assures that each elementary path is considered only once.

At some point, no vertices will be available for extension. We test for a circuit by seeing whether there is an arc connecting the last vertex of P to the first vertex. If there is, then a circuit is reported. In any case, vertex closure occurs unless there is only one vertex remaining in the path P.

Backtrack

Vertex closure consists of three steps:

1. Enter the last vertex of P into the list in H for the next to the last vertex in P.

2. Clear the list in H for the last vertex.

3. Shorten P by one arc by eliminating the last vertex.

(1) assures that the path extension just performed will not be repeated. (2) allows correct forward continuation from the last vertex if it is reached by a different path in the future.

The extension and backtracking continues until the path has been backed to the "initial" vertex 1. Then, the "initial" vertex is advanced. This means that the first vertex is incremented by one; H is cleared; and the extension process resumes. No paths, and thus circuits containing vertex 1 will be considered again. All circuits containing vertex 1 will have been found. The algorithm continues to extend paths and advance the "initial" vertex sequentially until P contains a path of one vertex, namely, vertex n. At this point, the algorithm terminates. All elementary circuits have been identified.

The algorithm discussed above, and the exact algorithm presented in ALGORITHM 2 is attributed to Tiernan [34]. Example 2 illustrates Tiernan's algorithm.

The author selected this algorithm by Tiernan [34] to introduce the fundamental idea behind the Search and Backtrack Methods because of its simplicity in exposition, and also because all the other Search and Backtrack Methods develop upon the main idea that was introduced by this algorithm. In Section 4.2, we will show how some slight modifications of Tiernan's algorithm solves a specific problem. In

ALGORITHM 2: An Algorithm for Enumerating all Elementary Circuits in the Graph (Tiernan)

B1. Initialize

Read N,G

$P \leftarrow 0$

$H \leftarrow 0$

$k \leftarrow 1$

$P(1) \leftarrow 1$

B2. Path Extension

Search G(P(k,j) for j = 1,2,...N such that the following three conditions are satisfied:

(1) $G(P(k),j) > P(1)$

(2) $G(P(k),j) \notin P$

(3) $G(P(k),j) \notin H(P(k),m)$, $m = 1,2,...N$.

If this j is found, extend the path,

$k \leftarrow k + 1$

$P(k) \leftarrow G(P(k - 1),j)$

go to B2.

If no j meets the above conditions, the path cannot be extended.

B3. Circuit Confirmation

If $P(1) \notin G(P(k),j)$, $j = 1,2,\ldots N$ then no circuit has been formed,

go to B4.

Otherwise a circuit is reported,

Print P.

B4. Vertex Closure

If $k = 1$, then all of the circuits containing vertex P(1) has been considered.

go to B5.

Otherwise,

$H(P(k),m) \leftarrow 0$, $m = 1,2,\ldots N$

For m such that $H(P(k - 1),m)$ is the leftmost zero in the

$P(K - 1)$ - the row of H,

$H(P(k - 1),m) \leftarrow P(k)$

$P(k) \leftarrow 0$

$k \leftarrow k - 1$

go to B2.

B5. Advance Initial Vertex

If $P(1) = N$ then

go to B6.

otherwise,

$P(1) \leftarrow P(1) + 1$

$k \leftarrow 1$

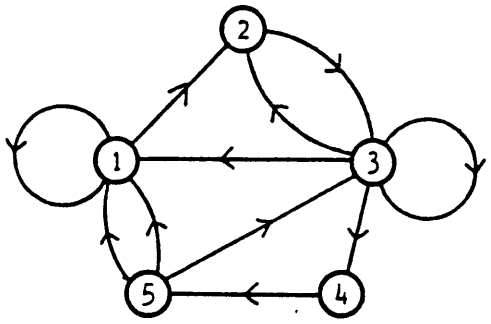
$H \leftarrow 0$

go to B2.

B6. Terminate

End of ALGORITHM 2

Example 2. Circuit Enumeration using Tiernan's algorithm presented in ALGORITHM 2.



$$G(V,\Gamma) = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 \end{bmatrix}$$

Note: The parallel arcs (5,1) have been replaced by a single arc (5,1).

P	ACTIONS ON ATTAINED PATH
1 0 0 0 0	B1. Initialization.
1 2 0 0 0	B2. Path extension.
1 2 3 0 0	B2. Path extension.
1 2 3 4 0	B2. Path extension.
1 2 3 4 5	B2. No path extension. B3. Circuit reported, (1 2 3 4 5 1). B4. Backtrack and vertex closure, H(4,1)<--5.
1 2 3 4 0	B2. No extension. B3. No circuit formed. B4. Clear last vertex, H(4,1)<--0, Backtrack and vertex closure H(3,1)<--4
1 2 3 0 0	B2. No extension. B3. Circuit reported, (1 2 3 1) B4. Clear the last vertex, H(3,1)<--0, Backtrack and vertex closure, H(2,1)<--3.
1 2 0 0 0	B2. No extension. B3. No circuit. B4. Clear last vertex, H(2,1)<--0, Backtrack and vertex closure H(1,1)<--2
1 0 0 0 0	B2. No extension. B3. Circuit reported, (1 1) B4. Cannot backtrack. B5. Advance vertex, i.e. set P(1) = 2, Clear H. Comment: All circuits containing vertex 1 have been enumerated.
2 0 0 0 0	B2. Path extension.
2 3 0 0 0	B2. Path extension.
2 3 4 0 0	B2. Path extension.
2 3 4 5 0	B2. No path extension. B3. No circuit found. B4. Backtrack and vertex closure, H(4,1)<--5.

2 3 4 0 0	B2. No extension. B3. No circuit formed. B4. Clear the last vertex, H(4,1)<--0, Backtrack and vertex closure, H(3,1)<--4.
2 3 0 0 0	B2. No path extension. B3. Circuit formed, (2 3 2) B4. Clear last vertex, H(3,1)<--0, Backtrack and vertex closure H(2,1)<--3.
2 0 0 0 0	B2. No path extension. B3. No circuit formed. B4. Cannot Backtrack. B5. Advance vertex, i.e. set P(1) = 3. Clear H. Comment: No circuit formed hereafter would contain vertices 1 or 2.
3 0 0 0 0	B2. Path extension.
3 4 0 0 0	B2. Path extension.
3 4 5 0 0	B2. No path extension. B3. Circuit reported, (3 4 5 3). B4. Backtrack and vertex closure, H(4,1)<--5.
3 4 0 0 0	B2. No path extension. B3. No circuit reported. B4. Clear last vertex H(4,1)<--0, Backtrack and vertex closure H(3,1)<--4.
3 0 0 0 0	B2. No path extension. B3. Circuit reported, (3 3). B4. Cannot Backtrack. B5. Advance vertex, i.e. set P(1) = 4. Clear H. Comment: No circuits formed hereafter would contain vertices 1, 2 or 3.
4 0 0 0 0	B2. Path extension.
4 5 0 0 0	B2. No path extension. B3. No circuit found. B4. Backtrack and vertex closure, H(4,1)<--5.
4 0 0 0 0	B2. No path extension. B3. No circuit formed. B4. Cannot Backtrack. B5. Advance vertex, i.e. set P(1) = 5, Clear H. Comment: No circuit formed hereafter would contain vertices 1, 2, 3 or 4.
5 0 0 0 0	B2. No extension possible. B3. No circuit formed. B4. Cannot Backtrack; all circuits containing P(1) = 5 have been found. B6. Since P(1) =5; Terminate. Comment: All circuits in the graph have been found.

Circuits founded are:-

2 1-circuits(self-loops), (1 1) and (3 3)
1 2-circuit, (2 3 2)
2 3-circuits, (1 2 3 1) and (3 4 5 3)
2 5-circuits, (1 2 3 4 5 1)

********** End of Example 2 **********

the discussions following, some similar methods are highlighted.

An example provided by Tarjan [32] shows the inefficiency of the above algorithm in the worst case. Weinblatt [35] provides an algorithm that examines each arc of the graph only once. He uses a recursive backtracking procedure to test combinations of subpaths from old circuits to see if they result in new ones. Tarjan [32], however, showed also that Weinblatt's algorithm does not have a running time polynomial to the number of circuits in the given graph.

Taijan [32] uses Tiernan's backtracking procedure but also uses a marking procedure to avoid unnecessary searches which help decrease the size of the subset of paths that need to be generated considerably. The running time of Tarjan's algorithm is shown to be polynomial to the number of circuits in the graph.

Johnson [18] and Szwarcfiter and Lauer [31] use improved pruning methods over Tarjan's [32]. Their algorithms have running times that are also polynomial to the number of circuits in the graph, but are an improvement over Tarjan's. In particular, Johnson's [18] algorithm is shown to be asymptotically fastest (for a large graph).

The algorithms of Char [5] and Chan and Chang [4] use the set of all permutations of vertices of the graph as the search space. Other algorithms using the Search and Backtrack Methods have also been presented by Berztiss [2], Roberts and Flores [29], Reed and Taijan [28] and Ehrenfeucht et al. [12].

2.3. Connection Matrix Methods

These methods make use of the properties of the connection matrix of a directed graph to generate elementary paths as vertex sequences. In our generation of elementary paths however, we would also be generating simple and non-simple paths.[3] The method eliminates all simple (but non elementary) and non-simple paths as soon as they are formed. It builds elementary paths one arc at a time and lists circuits for each cardinality in increasing order.

Before proceeding to discuss the method, a simple but important theorem is given. This theorem establishes the fundamental idea behind the Connection Matrix Method. We state the theorem for the adjacency matrix, but the idea can be easily extended to connection matrix since the difference between the adjacency and connection matrix is that the ij elements of the adjacency matrix are ones or zeros depending on whether arc ij exists whereas the ij elements in the connection matrix tells us the number of arcs from vertex i to j.

A directed graph can be represented as an n x n adjacency matrix, $A = [(a)_{ij}]$ where $(a)_{ij} = 1$ if arc ij exists and zero otherwise. Only self loops would appear as a non zero element in the diagonal of A. We state the theorem formally:

Theorem 3: The ij element of A^k is the number of paths of length k, or "k-paths" from i to j.

Proof of this theorem can be found in Narsingh Deo [10].

If a k-circuit (that is, of cardinality k) exists, there would be a non-zero element in the diagonal of the A^k matrix. However, a non zero element in the diagonal does not mean that a simple k circuit exists. The reason for this is that when we take the product of the adjacency matrices, infinite paths and/or circuits (that uses one or more arcs repeatedly) are formed. One could extend this result to the connection matrix, that is, the ij element for the matrix C_1^k (where C_1 is the connection matrix) equal to the number of paths of length k from vertex i to j.

A method to resolve the problem outlined above is suggested by Kamae [19]. This method breaks up circuit generation into three stages: (1) path enumeration, (2) flower enumeration and (3) circuit enumeration. We now proceed to discuss this method in more detail.

Define a connection matrix $C_1 = [(c_1)_{ij}]$ of a directed graph G such that the ij element, $(c_1)_{ij}$ equals to the number of arcs from i to j in G.[4] (Note that C_1 is similar to the adjacency matrix if there are no parallel arcs in G.) Next, define C_1' as a matrix where,

$$(c_1')_{ij} = (c_1)_{ij} \qquad \text{if } i \neq j$$
$$= 0 \qquad \text{for } i = j, \text{ where } i,j = 1,2,\ldots n$$

that is, C_1' is the same matrix as C_1 with self loops removed.

Then, let $C_2 = C_1' \cdot C_1' = (C_1')^2$ which means that each non zero element in C_2 indicates the number of 2-paths or 2-circuits. Next, let C_2' be the matrix C_2 with no 2-circuits, i.e.,

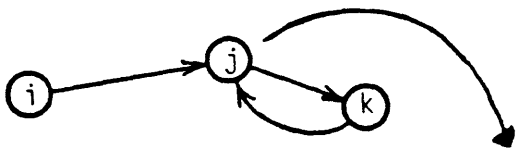
$$(c_2') = 0 \qquad \text{if } i = j$$
$$= (c_2)_{ij} \quad \text{otherwise}$$

then, consider the matrix multiplication $C_2' \cdot C_1'$ where the elements are:-

$$(c_2' \cdot c_1')_{ij} = \sum_{k=1}^{n} (c_2')_{ik} (c_1')_{kj}$$

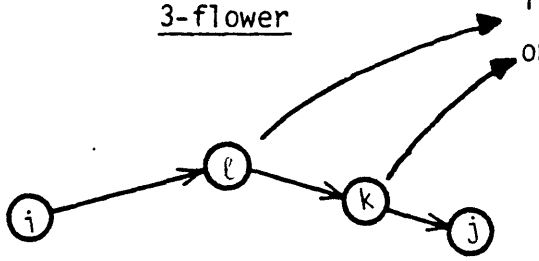
Given these elements, we will generate elementary paths, circuits and flowers. Suppose $i \neq j$, $i \neq k$ and $j \neq k$, we have two cases:

Case 1



3-flower

Case 2



intermediate points on 3-path from i to j.

3-path

Whenever i = k or j = k, $(c_2')_{ik} \cdot (c_1')_{kj} = 0$, thus i ≠ j implies that $(C_2' \cdot C_1')_{ij}$ is equal to the number of 3-paths and 3-flowers from i to j. Next, suppose i = j, then $(C_2' \cdot C_1')_{ij}$ equals to the number of 3-circuits containing i.

The method by Kamae [19], for more general cases is stated is stated in ALGORITHM 3.

Since all circuits must be of length lesser or equal to n, we need to compute only up to matrix Z_n.

We shall work on the same graph as used in section 2.2. to illustrate the method that was discussed in Example 3.

At this point, we can make several observations. Note that there is more than one way of obtaining the h-path connection matrix. We have merely illustrated one way of doing so in our example. Kamae's method chooses to eliminate non simple paths as the algorithm proceeds instead of sorting paths at various points in the algorithm. Due to this, Kamae's method is more favorable for computer implementation.

In general, different Connection Matrix Methods differ only in the way they avoid generating the arc or vertex sequences that can neither belong to paths nor circuits, i.e. non simple or infinite arc sequences.

Other algorithms that belong to this class can be found in Ponstein [26], Yau [37], Danielson [8] and Ardon and Malik [1]. In particular, Ardon and Malik reduce the storage bound to $O(n^2)$ by finding circuits, using Boolean reduction, to an expression which

ALGORITHM 3: Kamae's Algorithm for Generating All Elementary Circuits Using Connection Matrix Method.

1. Path Enumeration

Def. 1. The element of an h-path connection matrix, C_h, of G is defined by:

$(c_h)_{ij}$ = number of h-paths from i to j if $i \neq j$

= number of h elementary circuits containing i if $i = j$.

Def. 2. A proper h-path connection matrix C'_h of G is defined by:

$$(c'_h)_{ij} = (c_h)_{ij} \qquad \text{if } i \neq j$$

$$= 0 \qquad \text{if } i = j$$

Def. 3. Let L be the set of all (h-t)-circuits and μ be a particular (h-t)-circuit belonging to set L. Then, an (h-t) flower matrix $C_{h,t}$ of G is defined by,

$$C_{h,t} = \sum_{\mu \in L} C_\mu(h,t) , \qquad t \geq 1,\ h - t \geq 2$$

where $(c_{\mu(h,t)})_{ij}$ equals the number of t-paths from i to j which do not touch μ except at j if $i \notin \mu$ and $j \in \mu$, and

zero otherwise.

An (h,t)-flower is an h-flower which contains a t-path and an (h-t)-circuit with only one vertex in common, that is the joint. $C_{\mu(h,t)}$ then represents all (h,t)-flowers which contains μ, since a flower from i to j consists of a circuit μ containing j and a path from i to j not touching μ except at j. $C_{h,t}$ as defined above, is then the sum of $C_{\mu(h,t)}$ over all (h-t)-circuits, $(c_{h,t})_{ij}$ then equals to the number of (h,t)-flowers from i to j where j is the joint.

With this backtround, we are now able to state a theorem for C_h (The connection matrix which counts h-paths and h-circuits).

Theorem 4.

$$C_h = C_{h-1} \cdot C_1 - \sum_{t=1}^{h-2} c_{h,t} \qquad h \geq 3$$

$$C_h = C'_{h-1} \cdot C'_1 \qquad h = 2$$

$\sum_{t=1}^{h-2} C_{h,t}$ denotes the sum of the number of (h-t)-flowers where $t = 1,\ldots h-2$. We are essentially removing all flowers each time we try to find C_h. Note that for h = 2, no flowers can be formed therefore the term $\sum_{t=1}^{h-2} C_{h,t}$ is not needed.

2. Flower Enumeration

Def. 4.

$$(D_1^{\mu})_{qr} = 0 \qquad \forall\, q \in \mu$$

$$= (c'_1)_{qr} \qquad \text{otherwise .}$$

that is, D_1^{μ} is a matrix of the graph G obtained from the connection matrix by deleting all edges incident from vertices contained in circuit μ. D_h^{μ}, $D_h'^{\mu}$ etc., are defined in parallel with C_h, C_h' etc. So with theorem 4 for connection matrices, we have,

Theorem 5:

$$D_t^{\mu} = D_{t-1}'^{\mu} \cdot D_1'^{\mu} - \sum_{s=1}^{t-2} D_{t,s} \qquad t \geq 3$$

$$D_t^{\mu} = D_{t-1}'^{\mu} \cdot D_1'^{\mu} \qquad t = 2$$

As in theorem 4, $\sum_{s=1}^{t-2} D_{t,s}$ denotes the flowers of length t, with circuits of length t-1 and less (but not lesser than 2). We next state another theorem which will be useful for obtaining the number of flowers.

Theorem 6: Recall that μ is a (h-t) circuit,

$$(D_t^{\mu})_{ij} = (c_{\mu(h,t)})_{ij} \qquad \text{if } i \notin \mu \text{ and } j \in \mu$$

$$= 0 \qquad \text{otherwise}$$

From definition 4, for $i \notin \mu$ and $j \in \mu$, $(D_t^{\mu})_{ij}$ is the number of t-paths from i to j, not touching μ except at j (since j has outdegree 0 for the graph defined by D_1^{μ}). Note that this corresponds to the definition of $(c_{\mu(h,t)})_{ij}$. (See Def. 3). Hence, $(c_{\mu(h,t)})_{ij}$ can be determined from D_t^{μ}. This is desirable since by applying definition 5 and theorems 5 and 6 repeatedly, we can obtain D_t^{μ}.

3. Circuit Generation

After obtaining c_h, we would like to be able to list the circuits as vertex sequences. We know that all h-circuits appear as a non zero element in the diagonal of C_h. The following definition will help us.

Def. 5. A h-circuit matrix Z_h is defined as,

$$(z_h)_{ij} = (c'_{h-1})_{ji} \; (c'_1)_{ij} \qquad 1 \leq i, \; j \leq n$$

Notice that since $(c'_{h-1})_{ji}$ equals to the number of (h-1) paths from j to i, and $(c'_1)_{ij}$ equals the number of arcs from i to j, $(z_h)_{ij}$ is the number of h-circuits which contains an arc from i to j. The difference between $(z_h)_{ij}$ and $(c_h)_{ij}$ for i = j, is that the former provides us with arcs belonging to a h-circuit (that is, a starting point for listing our circuit, whereas the latter just tells us the number of h-circuits containing a certain vertex.

Having obtained Z_h, we list the h-circuits by making the first non-zero element in the first non zero row, and list the first arc belonging to the h-circuit. For example, pq, observe that p represents the row and q, the column in which the non-zero element appears. Next, we proceed to the q^{th} row; search for the first (left most) non zero element, and list the next arc in the h-circuit by reading off the corresponding row and column, for instance qr. We proceed in this manner, building an arc at a time until we return to the initial vertex p. Note that we are guaranteed to return to p after connecting

h arcs. The h-circuit is then pqr...p. In some instances, there might exist more than one h-circuit or more than one that uses the same arc many times. It is then necessary for us to do two things:-

1. After each h-circuit has been listed, we remove all arcs of the h-circuit from the Z_h matrix. This is accomplished as follows:-

$$(z'_h)_{ij} = (z_h)_{ij} - 1 \qquad \forall \text{ ij belonging to h-circuit listed}$$

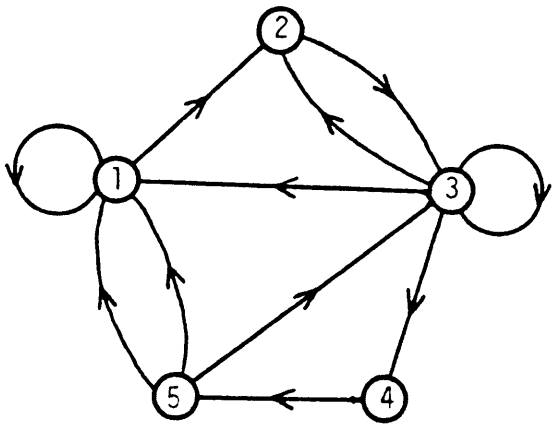
$$= (z_h)_{ij} \qquad \text{otherwise}$$

The resultant matrix Z'_h contains the remainder of the h-circuits of the graph.

2. We return to our procedure for listing h-circuits until $(z'_h)_{ij} = 0$, for all i and j.

---------- End of ALGORITHM 3 ----------

Example 3. Kamae's Algorithm Applied to Circuit Enumeration

	Results	Comments
1.	$C = C_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 \end{bmatrix}$	
	From C_1, two 1-circuits, 1,1,3,3 are read off the diagonal	By definition 1.
2.	$C_1' = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 \end{bmatrix}$	By definition 2.
3.	$C_2 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 3 & 0 & 1 & 0 \end{bmatrix}$	$C_2 = (C_1')^2$ By definition 1 and theorem 4.

Results	Comments
4. $Z_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ From Z_2, a 2-circuit 232 is obtained.	From definition 5, $(z_2)_{ij} = (c_1')_{ji} \cdot (c_1')_{ij}$ e.g. $(z_2)_{23} = (c_1')_{32} \cdot (c_1')_{23} = 1$
5. $C_2' = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 3 & 0 & 1 & 0 \end{bmatrix}$	C_2' is obtained from definition 2. At this point, we have a one 2-circuit. Let μ = 2-circuit 232.
6. $D_1^{\mu} = D_1'^{\mu} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 \end{bmatrix}$	We obtain D_1^{μ} by eliminating rows 2 and 3 from C_1 i.e. by definition 4.

Results	Comments
7. $C_{3,1} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$	From $D_1'^{\mu}$, we obtain by keeping columns 2 and 3, and setting the remainder columns to zero. (Theorem 6) Note: $C_{\mu(3,1)} = C_{3,1}$ since there is only one 2-circuit.
8. $C_2' \cdot C_1' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 2 & 0 & 0 \\ 1 & 3 & 0 & 1 & 0 \\ 0 & 1 & 3 & 0 & 1 \end{bmatrix}$	
9. $C_3 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 2 & 0 & 0 \\ 1 & 3 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 & 1 \end{bmatrix}$	From theorem 4, $C_3 = C_2' \cdot C_1' - C_{3,1}$

Results	Comments
10. $Z_3 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$	$(z_3)_{ij} = (c_2')_{ji} \cdot (c_1')_{ij}$ i.e. Definition 5.
From z, we obtain the 3-circuits 1231 and 3453.	Let ν be the 3-circuit 1231 and λ be the 3-circuit 3453.
11. $C_3' = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \end{bmatrix}$	By definition 2.
12. $D_1^{\nu} = D_1'^{\nu} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 \end{bmatrix}$	Since ν is the 3-circuit 1231, we obtain D_1^{ν} by removing rows 1,2,3 from C_1. (From theorem 6.)

Results	Comments
13. $C_{\nu(4,1)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \end{bmatrix}$	We obtain $c_{\nu(4,1)}$ by keeping columns 1,2,3, from $D_1'^{\nu}$.
14. $D_1^{\lambda} = D_1'^{\lambda} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	Since λ = 3-circuit 3453, we obtain D_1^{λ} by removing rows 3,4,5 from C_1 (Again, Theorem 6)
15. $C_{\lambda(4,1)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	From $D_1'^{\lambda}$ we obtain $C_{\lambda(4,1)}$ by keeping columns 3,4,5 from $D_1'^{\lambda}$.
16. $C_{4,1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \end{bmatrix}$	Hence, $C_{4,1} = C_{\nu(4,1)} + C_{\lambda(4,1)}$ i.e. from definition 3 and theorem 6.

Results	Comments
17. $D_2^{\mu} = D_2'^{\mu} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$	From D_1^{μ}, we obtain $D_2^{\mu} = (D_1^{\mu}) \cdot (D_1^{\mu})$ (Theorem 5)
18. $C_{4,2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix}$	Hence, we can find $C_{4,2}$ from $D_2'^{\mu}$ by returning columns 2 and 3.
19. $C_3' \cdot C_1' = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 & 0 \\ 2 & 2 & 1 & 2 & 0 \end{bmatrix}$	
20. $C_4 = C_4' = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix}$	From what has been obtained, we get, $C_4 = C_3' \cdot C_1' - C_{4,1} - C_{4,2}$ (Theorem 4) $C_4 = C_4'$ since diagonal elements of C_4 are zero.

Results	Comments
21. $C_{5,1} = [0]$	Since there are no 4-circuits.
22. $D_2^{\nu} = D_2'^{\nu} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	Now, for 3-circuit ν, $D_2^{\nu} = D_1^{\nu} \cdot D_1^{\nu}$ (Theorem 5)
23. $D_2^{\lambda} = D_2'^{\lambda} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	And, for 3-circuit λ, $D_2^{\lambda} = D_1^{\lambda} \cdot D_1^{\lambda}$ (Theorem 5)
24. $C_{5,2} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	The same way $C_{4,1}$ was obtained. Note however that we are doing this for all 3-circuits now, i.e. ν and λ (left as an exercise). (Theorem 6 and Definition 3)
25. $D_3^{\mu} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	From $D_1'^{\mu}$ and $D_2'^{\mu}$, $D_3^{\mu} = D_2'^{\mu} \cdot D_1'^{\mu} - D_{3,1}^{\mu}$ (Theorem 5) Note: $D_{3,1}^{\mu} = 0$ because D_2^{μ} has no diagonal elements or no 2-circuit. $\therefore D_3^{\mu} = D_2'^{\mu} \cdot D_1'^{\mu}$

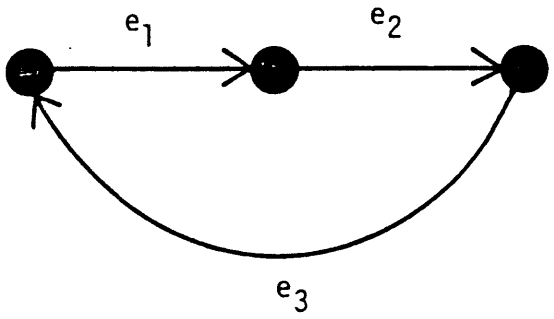
Results	Comments
26. $C_{5,3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$	From D_3^{μ}, we obtain $C_{5,3}$ by keeping columns 2 and 3 corresponding to 2-circuit μ. (μ = 2-circuit 2,3,2) (Theorem 6)
27. $C_5 = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$	$C_5 = C_4' \cdot C_1' - \sum_{t=1}^{3} C_{5,t}$ (By Theorem 4)
28. $Z_5 = \begin{bmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix}$ There are two 5-circuits, both of which are identical, 123451.	By definition 5, $(z_5)_{ij} = (c_4')_{ji} \cdot (c_1')_{ij}$

********** End of Example 3 **********

results from the permanent expansion of matrix M, where M = C + I and C is the variable adjacency matrix and I, the identity matrix. A new expansion, the "pseudopermanent," is defined by which the set of circuits can be formed directly. An extension of the method to find Hamiltonian circuits is also included. The method of Ardon and Malik offers improvements over all the other Connection Matrix Methods since the storage requirement is $O(n^2)$ as opposed to $O(n(const)^n)$ for the others. We will mention more about the comparison between algorithms in Section 3.

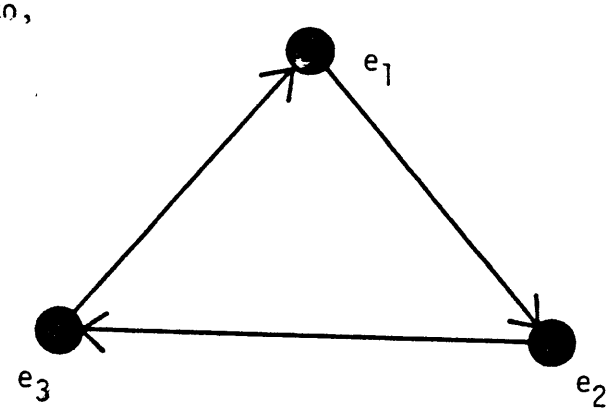
2.4. Directed Graph Transformation Method

A directed graph can be transformed into a "line graph" where the properties of the "line graph" are useful for the purpose of circuit enumeration. Given a directed graph G(V,E), where $e_i \in E$, the associated "line graph", $\Omega(G)$, is a graph where each arc in G represents a vertex in Ω and each two adjacent arcs in E form a vertex sequence connected by an arc in $\Omega(G)$, that is:



G

transforms to,



$\Omega(G)$

The arc set of G is then the vertex set of Ω. Each p-path of G will correspond to a (p-1) path in Ω. However, each p-circuit of G will correspond to a p-circuit in Ω. There is a one-to-one correspondence between circuits in G and Ω. (See Cartwright and Gleason [3] for proof.) The elegance of this method is that we will be able to enumerate and delete the circuits in Ω without disrupting the cyclic nature of the original graph.

The algorithm lists all self loop first and eliminates them for G. For G, find $\Omega(G)$ and enumerate, then delete all arcs which are members of 2-circuits in $\Omega(G)$. Let the resulting graph subgraph be G^1. We then proceed to find $\Omega(G^1)$, enumerate, then delete all 3-circuits. Note that $\Omega(G^1)$ has only circuits of length greater or equal to 3. Call the resulting subgraphs G^2, find $\Omega(G^2)$, and so on ..., until $\Omega(G^P)$ for $p \leq n-2$ is empty. Example 4 will illustrate the method more clearly.

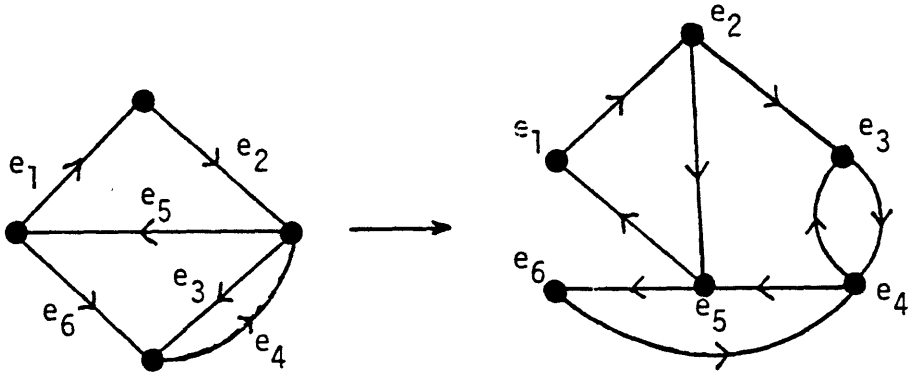
The method outlined above relied on the one-to-one correspondence between circuits in G and $\Omega(G)$. It allows us to remove circuits as they are formed until eventually none are left, at which point the algorithm terminates. Observe that just as is the case of the Connection Matrix Methods, all circuits of identical cardinality are found simultaneously.

This method is well suited if the majority of the circuits in the graph we are studying have a small cardinality. Alternatively, we might be interested in circuits of a certain (small) cardinality, or circuits that do not exceed a certain cardinality. The reason for this is that whenever circuits are enumerated, they are also removed, thus the size of the graph is quickly reduced, and convergence might be faster as well.[6] No other method allows us to remove circuits from the graph and test the resulting subgraph for circuits.

Example 4. Enumeration of circuits using Directed Graph Transformation Method.

Iteration 1.

Convert G to $\Omega(G)$



G $\Omega(G)$

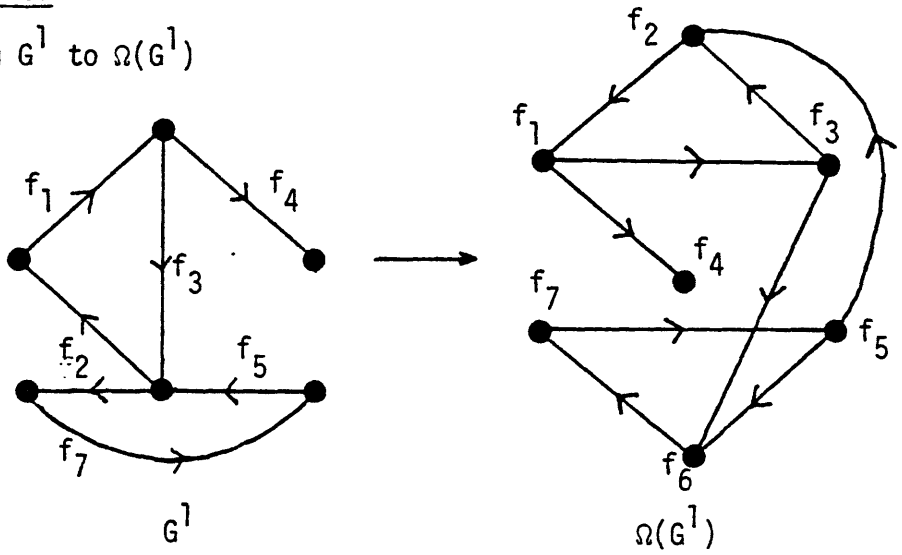
List 2-circuit: $e_3\ e_4\ e_3$ in $\Omega(G)$.

Delete arcs (e_3, e_4) and (e_4, e_3) from $\Omega(G)$.

The resulting subgraph is known as G^1.

Iteration 2.

Transform G^1 to $\Omega(G^1)$



G^1 $\Omega(G^1)$

List 3-circuits: $f_1\ f_3\ f_2\ f_1$ $(e_1\ e_2\ e_5\ e_1)$ and

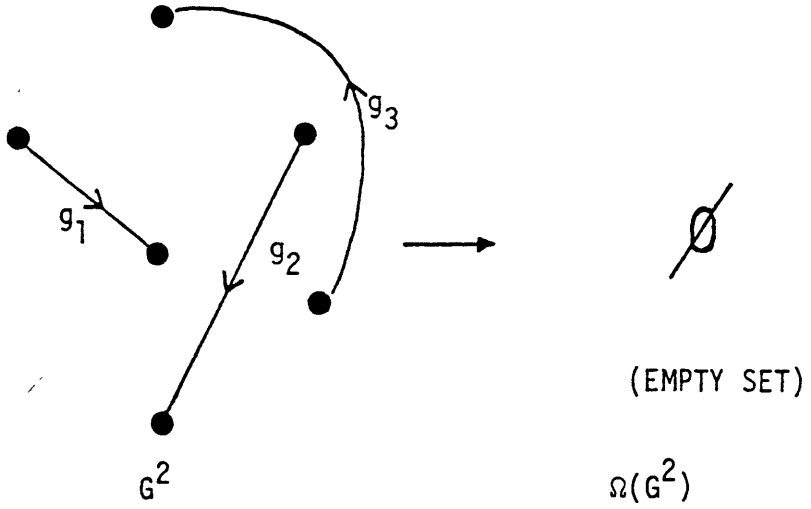
$f_5\ f_6\ f_7\ f_5$ $(e_4\ e_5\ e_6\ e_4)$ in $\Omega(G^1)$

Delete arcs $(f_1\ ,\ f_3)$, $(f_3\ ,\ f_2)$, $(f_2\ ,\ f_1)$, $(f_5\ ,\ f_6)$, $(f_6\ ,\ f_7)$ and $(f_7\ ,\ f_5)$ from $\Omega(G^1)$

The resulting subgraph is now known as G^2.

Iteration 3.

Transform G^2 to $\Omega(G^2)$



No 4-circuits are found.

$\Omega(G^2) = \emptyset$. This means that all the circuits have been enumerated.

Circuits found are:-

$e_3\ e_4\ e_3$

$e_1\ e_2\ e_5\ e_1$

$e_4\ e_5\ e_6\ e_4$

********** End of Example 4 **********

Cartwright and Gleason [3] have proposed a way of listing circuits from the "line graph," after each transformation (and reduction). However, we could also use the method outlined in Section 2.3 (Kamae's Method) to enumerate circuits of a given cardinality from the "line graph." The problem still remains with manipulating such huge sparse matrices, which would incur huge storage and elaborate computations.

The transformation from a directed graph to a "line graph" involves a simple logical relationship which could be further exploited. Instead of representing the graph as an adjacency matrix, it could also be represented by arc listing, or other methods that are less storage incurring.[7] Operations upon arc listings remain difficult and unexplored and point to possible areas of research.

2.5. Obtaining All Simple Circuits from the Set of Elementary Circuits

In the preceding subsections, we have confined ourselves to enumerating elementary circuits. In this subsection, we shall present an algorithm for generating all simple circuits, given the set of elementary circuits obtained by any one of the methods discussed in Sections 2.2 to 2.4. This algorithm is applicable to undirected graphs as well, where we are interested in finding all the simple cycles. We have taken the initiative here for two main reasons:

i. The enumeration of all simple circuits from the original graph is complex and as a result inefficient. Furthermore, the number of elementary circuits found in a graph would help us determine whether it would be wise to proceed with the generation of all simple circuits. This is also because the algorithm that we are about to propose has a worst case time bound related to the number of elementary circuits.

ii. In some problems, the set of all simple circuits corresponds to a feasible set of solutions for those problems. Additional constraints placed on this set of feasible solutions yield the optimum solution, if one exists. A typical problem would be to list the cheapest simple circuit for each given cardinality. After we have obtained this set of simple circuits, we could then use it for dispatching of vehicles. Other examples are best shown by figs. 2.1 and 2.2.

Figure 2.1 shows the relationships between simple circuits, elementary circuits and Hamiltonian circuit. It also shows that all solutions to the travelling salesman problem (TSP) are Hamiltonian circuits. In fig. 2.2 , we notice that if an Eulerian circuit exists, it corresponds to the solution for the Chinese postman problem (CPP). An Eulerian circuit is a simple circuit that covers all the arcs in the graph. Note that the solution for the CPP does not have to be simple circuits though. (TSP and CPP are discussed in more detail in Christofides [7].)

fig. 2.1 Relationship between Elementary and Simple Circuits

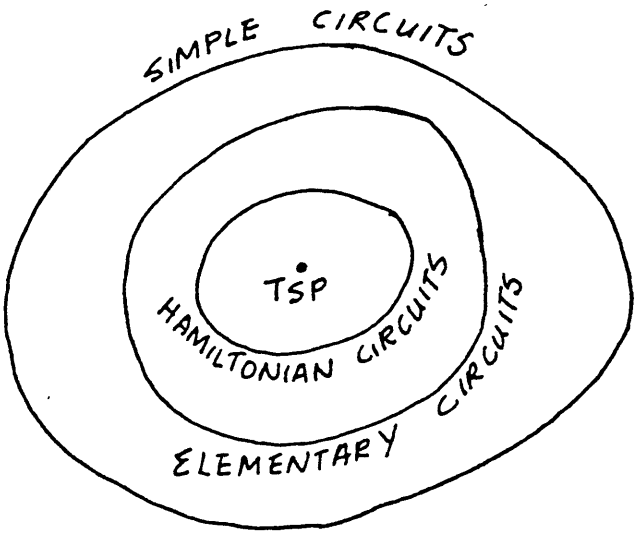
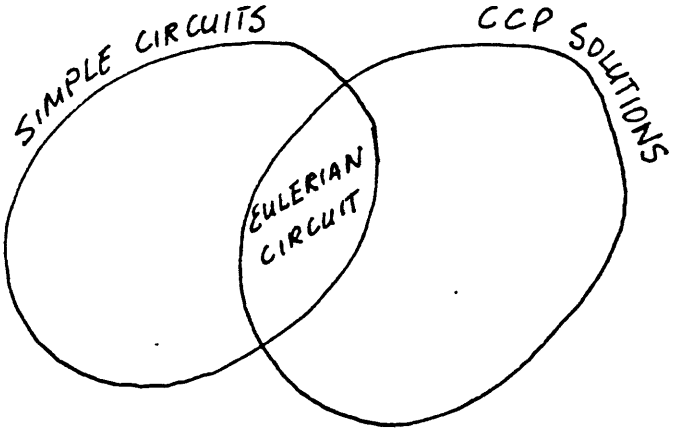


fig. 2.2 Simple, Eulerian Circuits and CCP



The algorithm which we will present shortly is not meant to solve some of these problems since other methods are available which are more efficient, but rather to illustrate the scope of this endeavor.

The following is an outline of the method:

Let there be q elementary circuits in a given graph defined by S_i, where i = 1,2,...q. S_i is an e-triple row vector where the j^{th} entry is 1 if the arc j is contained in the circuit, and zero otherwise. We define $S_i \cap S_k = \phi$ for $1 \leq i, k \leq q$ if S_i and S_k contains no arc in common, otherwise $S_i \cap S_k \neq \phi$. If $S_i \cap S_k \neq 0$, then the $S_i + S_k$ will not form a simple cycle. For example, if $S_1 = (1,0,0,1)$, $S_2 = (1,0,0,0)$, $S_3 = (0,1,1,0)$, then $S_1 \cap S_2 \neq \circ$ and $S_1 \cap S_3$, $S_2 \cap S_3 = \phi$. (One way to find out is to see if the addition of two vectors contain any element with value greater than one.)

Now, if $S_i \cap S_k = \phi$, then $S_i + S_k$ would be an arc disjoint union of elementary circuits or a simple but non-elementary circuit. Note however, that if $P \cap S_i = \phi$, where P is a general circuit which may have more than one circuit component. Specifically, $P \varepsilon M$, where M = {the set of all arc disjoint unions of elementary circuits or an arc-disjoint union of elementary and simple circuits, or an arc-disjoint union of simple circuits, or a simple but non-elementary circuit}, then $P + S_i \varepsilon M$. To distinguish whether $P + S_i$ is a simple but non-elementary circuit, a few definitions are necessary.

We define $V(S_i)$ as an n-triple row vector (where n is the total number of vertices in the graph) of circuit S_i where the j^{th} entry

(column) is 1 if vertex j belongs to circuit S_i and zero otherwise. Next, we define the operation $\odot$, where: $V(P) \odot V(S_i) \triangleq$ sum of all positive elements of $\{(V(P) + V(S_i)) - \vec{1}\}$; $\vec{1}$ is an n-triple vector of ones. This operation defines the number of vertex intersection at P and S_i. We denote also the number of distinct element circuits in P as x_p. For example, let $V(S_1) = (1,1,0,1,1)$, $V(S_2) = (0,1,0,0,0)$, $V(S_3) = (1,1,0,1,0)$. If P is $S_1 + S_3$, then $V(P) = V(S_1) + V(S_3)$,[8] i.e. $x_p = 2$ and $S_i = S_2$, then,

$$\{(V(P) + V(S_2) - \vec{1}\} = (2,3,0,2,1) - (1,1,1,1,1) = (1,2,-1,1,0)$$

and

$$V(P) \odot V(S_2) = 1 + 2 + 1 = 4 .$$

Note that $V(P) \odot V(S_2) \geq x_p$.

Now, given that $P \cap S_i = \phi$, if also $V(P) \odot V(S_i) \geq x_p$, then $P + S_i$ forms a simple but non-elementary circuit. We claim here that if x_p circuits are joined together such that they do not share any arc in common and they meet at least x_p times, then a simple but non-elementary circuit is formed.

If, however, $P \cap S_i = \phi$ and $V(P) \odot V(S_i) < x_p$, we must not discard $P + S_i$ from further comparisons, since the addition of $(P + S_i)$ and another elementary circuit might form a simple but

non-elementary circuit. (We store all these in set M in the following ALGORITHM 4.)

But if at any point $P \cap S_i \neq \phi$, then any further comparisons, that is, additions with $P + S_i$ can be eliminated since these combinations can never form a simple circuit. This elimination rule reduces the possible outcomes we need to consider, which otherwise would be enormous.

The actual algorithm is presented in ALGORITHM 4.

In order to reduce the number of combinations that need to be considered, it is better to order the circuits in the set of elementary circuits with decreasing cardinality, that is, S_1 is the elementary circuit with the highest cardinality, followed by S_2, S_3, etc. As usual, we include an example to complete our illustration. We refer to the same graph used previously, and reproduced here for convenience in Example 5.

It is important to mention here that in the worst-case the algorithm requires $2^q - 1$ combinations, where q is the number of elementary circuits. However, the worst case is highly unlikely since this would mean that each combination would result in a distinct simple circuit. In that case, by the ordering procedure that we have recommended, the number of combinations that need to be considered can be reduced. The amount of reduction would depend on the nature of the set of elementary circuits. Even then, this algorithm requires alot of computing time. We make every effort to

ALGORITHM 4. Algorithm to Find all the Simple Circuits in a Graph Given the Set of Elementary Circuits.

Let the set of elementary circuits be:-

$$S_i \; ; \; i = 1,2,\ldots q \; .$$

Initialize: 1 Set $L = \{S_1\}$, $M = \{S_1\}$ $i = 2$, $I = \phi$, $J = \phi$

Separate: 2 For each P in M

if $P \cap S_i = \phi$, place $P + S_i$ in I; (P,S_i) in J

Otherwise, continue until all p in M are considered.

3 $M \leftarrow M \cup I \cup S_i$

Test for : Simple Circuit 4 For each pair (P,S_i) in J

if $V(P) \odot V(S_i) < x_p$, set $I \leftarrow I - \{P + S_i\}$

Otherwise continue until all pairs in J are considered.

5 $L \leftarrow L \cup S_i \cup I$

6 Reset $I \leftarrow \phi$; Reset $J \leftarrow \phi$

7 Set $i = i + 1$. If $i \leq q$ go to 2

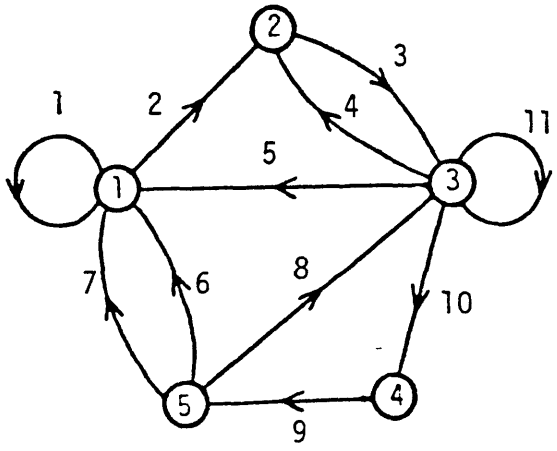
otherwise, stop.

L contains all simple circuits in the graph.

End of ALGORITHM 4

Example 5. Generating All Simple Circuits From the Set of Elementary Circuits

Given,



and,

Elementary Circuits	Arcs											
	1	2	3	4	5	6	7	8	9	10	11	12
S_1	0	1	1	0	0	1	0	0	1	1	0	0
S_2	0	1	1	0	0	0	1	0	1	1	0	0
S_3	0	1	1	0	1	0	0	0	0	0	0	0
S_4	0	0	0	0	0	0	0	1	1	1	0	0
S_5	1	0	0	0	0	0	0	0	0	0	0	0
S_6	0	0	0	0	0	0	0	0	0	0	1	0

	Vertices				
	1	2	3	4	5
$V(S_1)$	1	1	1	1	1
$V(S_2)$	1	1	1	1	1
$V(S_3)$	1	1	1	0	0
$V(S_4)$	0	0	1	1	1
$V(S_5)$	1	0	0	0	0
$V(S_6)$	0	0	1	0	0

We have S_i, $i = 1,2,\ldots 6$

1. Set $L = \{S_1\}$ $M = \{S_1\}$ $i = 2$, $I = \phi$, $J = \phi$.

---

2. $S_1 \cap S_2 \neq \phi$, $M \leftarrow \{S_1,S_2\}$, $L \leftarrow \{S_1,S_2\}$, $i = 3$

---

3. $S_1 \cap S_3 \neq \phi$, $S_2 \cap S_3 \neq \phi$, $M \leftarrow \{S_1,S_2,S_3\}$, $L \leftarrow \{S_1,S_2,S_3\}$, $i = 4$

---

4. $S_1 \cap S_4 \neq \phi$, $S_2 \cap S_4 \neq \phi$, S_3 $S_4 = \phi$, $I \leftarrow \{S_3 + S_4\}$; $J \leftarrow \{(S_3,S_4)\}$

   $M \leftarrow \{S_1,S_2,S_3,S_4,S_3 + S_4\}$, since $V(S_3) \odot V(S_4) = 1 = x_p$,

   $L \leftarrow \{S_1,S_2,S_3,S_4,S_3 + S_4\}$.

   $i = 5$

---

5. $S_1 \cap S_5 = \phi$, $S_2 \cap S_5 = \phi$, $S_3 \cap S_5 = \phi$, $S_4 \cap S_5 = \phi$, $(S_3 + S_4) \cap S_5 = \phi$

$I \leftarrow \{S_1 + S_5, S_2 + S_5, S_3 + S_5, S_4 + S_5, S_3 + S_4 + S_5\}$,

$J \leftarrow \{(S_1, S_5), (S_2, S_5), (S_3, S_5), (S_4, S_5), (S_3 + S_4, S_5)\}$,

$M \leftarrow \{S_1, S_2, S_3, S_4, S_3 + S_4, S_5, S_1 + S_5, S_2 + S_5, S_3 + S_5, S_4 + S_5, S_3 + S_4 + S_5\}$

Since, all except $V(S_4) \odot V(S_5) < x_p$ = (2 in this case),

$I \leftarrow \{S_1 + S_5, S_2 + S_5, S_3 + S_5, S_3 + S_4 + S_5\}$

$L \leftarrow \{S_1, S_2, S_3, S_4, S_3 + S_4, S_5, S_1 + S_5, S_2 + S_5, S_3 + S_5, S_3 + S_4 + S_5\}$. $i = 6$.

---

6. $S_1 \cap S_6 = \phi$, $S_2 \cap S_6 = \phi$, $S_3 \cap S_6 = \phi$, $S_4 \cap S_6 = \phi$, $S_5 \cap S_6 = \phi$,

$(S_3 + S_4) \cap S_6 = \phi$, $(S_1 + S_5) \cap S_6 = \phi$, $(S_2 + S_5) \cap S_6 = \phi$,

$(S_3 + S_5) \cap S_6 = \phi$, $(S_4 + S_5) \cap S_6 = \phi$, $(S_3 + S_4 + S_5) \cap S_6 = \phi$.

$I \leftarrow \{S_1 + S_6, S_2 + S_6, S_3 + S_6, S_4 + S_6, S_5 + S_6, S_3 + S_4 + S_6, S_1 + S_5 + S_6, S_2 + S_5 + S_6,$

$S_3 + S_5 + S_6, S_4 + S_5 + S_6, S_3 + S_4 + S_5 + S_6\}$

$J \leftarrow \{(S_1, S_6), (S_2, S_6), (S_3, S_6), (S_4, S_6), (S_5, S_6), (S_3 + S_4, S_6), (S_1 + S_5, S_6),$

$(S_2 + S_5, S_6), (S_3 + S_5, S_6), (S_4 + S_5, S_6), (S_3 + S_4 + S_5, S_6)\}$.

$M \leftarrow \{S_1, S_2, S_3, S_4, S_3 + S_4, S_5, S_1 + S_5, S_2 + S_5, S_3 + S_5, S_4 + S_5, S_3 + S_4 + S_5,$

$S_1 + S_6, S_2 + S_6, S_3 + S_6, S_4 + S_6, S_5 + S_6, S_3 + S_4 + S_6, S_1 + S_5 + S_6,$

$S_2 + S_5 + S_6, S_3 + S_5 + S_6, S_4 + S_5 + S_6, S_3 + S_4 + S_5 + S_6\}$

Since all except $V(5) \odot V(6)$ and $(V(4) + V(5)) \odot V(6)$ is not less than x_p,

$I \leftarrow I - \{S_5 + S_6, S_2 + S_5 + S_6\}$.

$L \leftarrow \{S_1, S_2, S_3, S_4, S_3 + S_4, S_5, S_1 + S_5, S_2 + S_5, S_3 + S_5, S_3 + S_4 + S_5, S_1 + S_6,$

$S_2 + S_6, S_3 + S_6, S_4 + S_6, S_3 + S_4 + S_6, S_1 + S_5 + S_6, S_2 + S_5 + S_6,$

$S_3 + S_5 + S_6, S_3 + S_4 + S_5 + S_6\}$. $i = 7$. Since $i > q$. Stop.

L contains all the simple circuits of the graph.

********** End of Example 5 **********

reduce the kind of operations to include simple additions only.

Slight modifications of the algorithm could very effectively search for the existence of an Eulerian circuit if one exists, or to find the set of simple circuits that contains a specific number of elementary circuits. Restrictions on the kind of simple circuits we would be interested in is what makes the algorithm more appealing.

In the next subsection, we look at generation of elementary cycles and circuits in a planar graph, and look at some complications that arise.

2.6. Methods for Generating all Cycles and Circuits in a Planar Graph

The methods discussed thus far apply to all graphs. This subsection deals with a particular kind of graph - planar graphs. There are certain properties that planar graphs have and this subsection presents another perspective on circuit enumeration using these properties.

Before proceeding, we would like to know how to recognize a planar graph.[9]

A graph is planar if and only if it can be mapped onto the surface of a sphere such that no two edges or arcs meet except at the vertex or vertices, with which the edges or arcs are incident. Fig. 2.3 shows some regular polygons and their representations as planar graphs in fig. 2.4 . A face of a planar graph is an area of the plane bounded by edges called contours which contains no edges, arcs, or vertices. A finite face is a face where the area

fig. 2.3 Some regular polygons

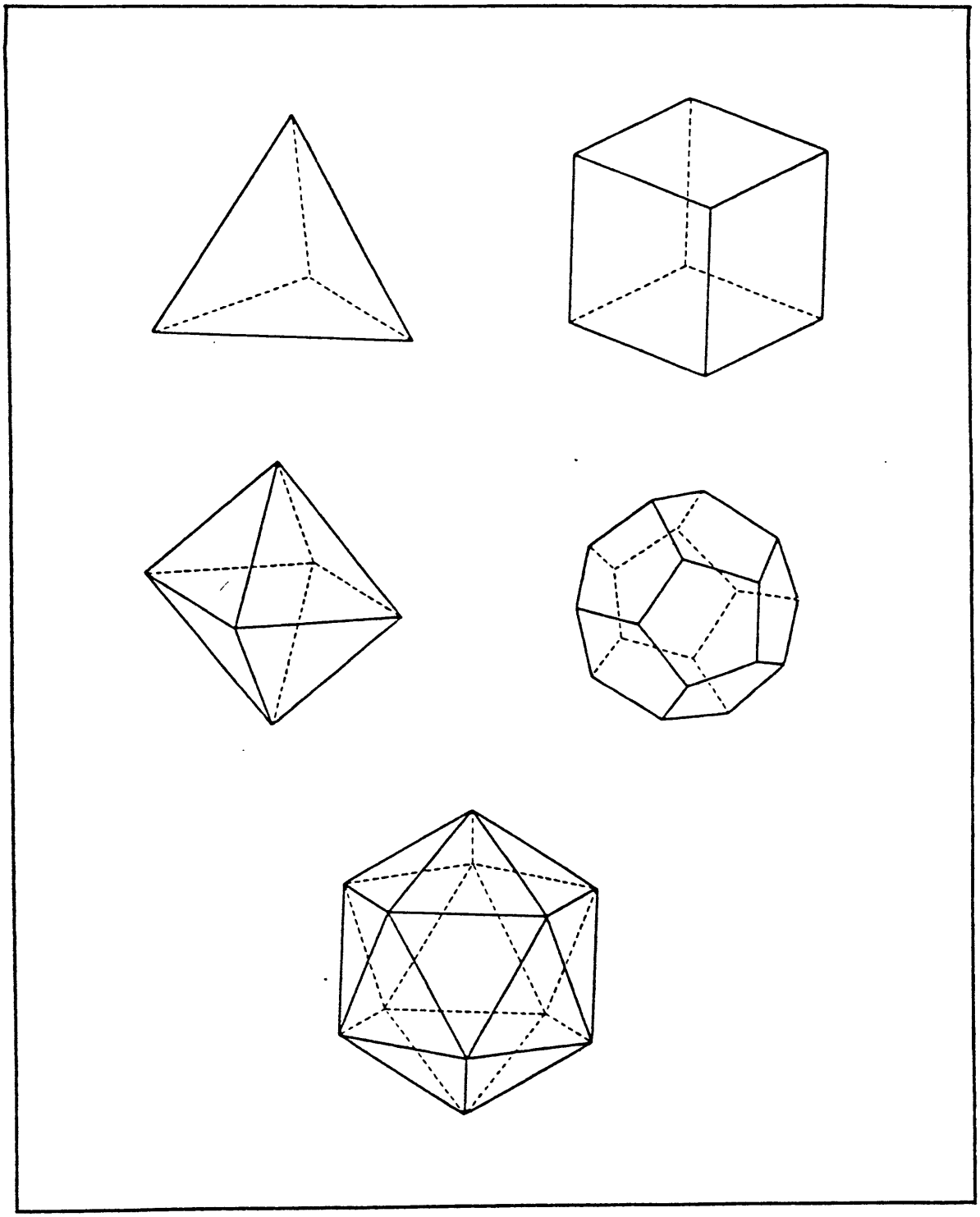
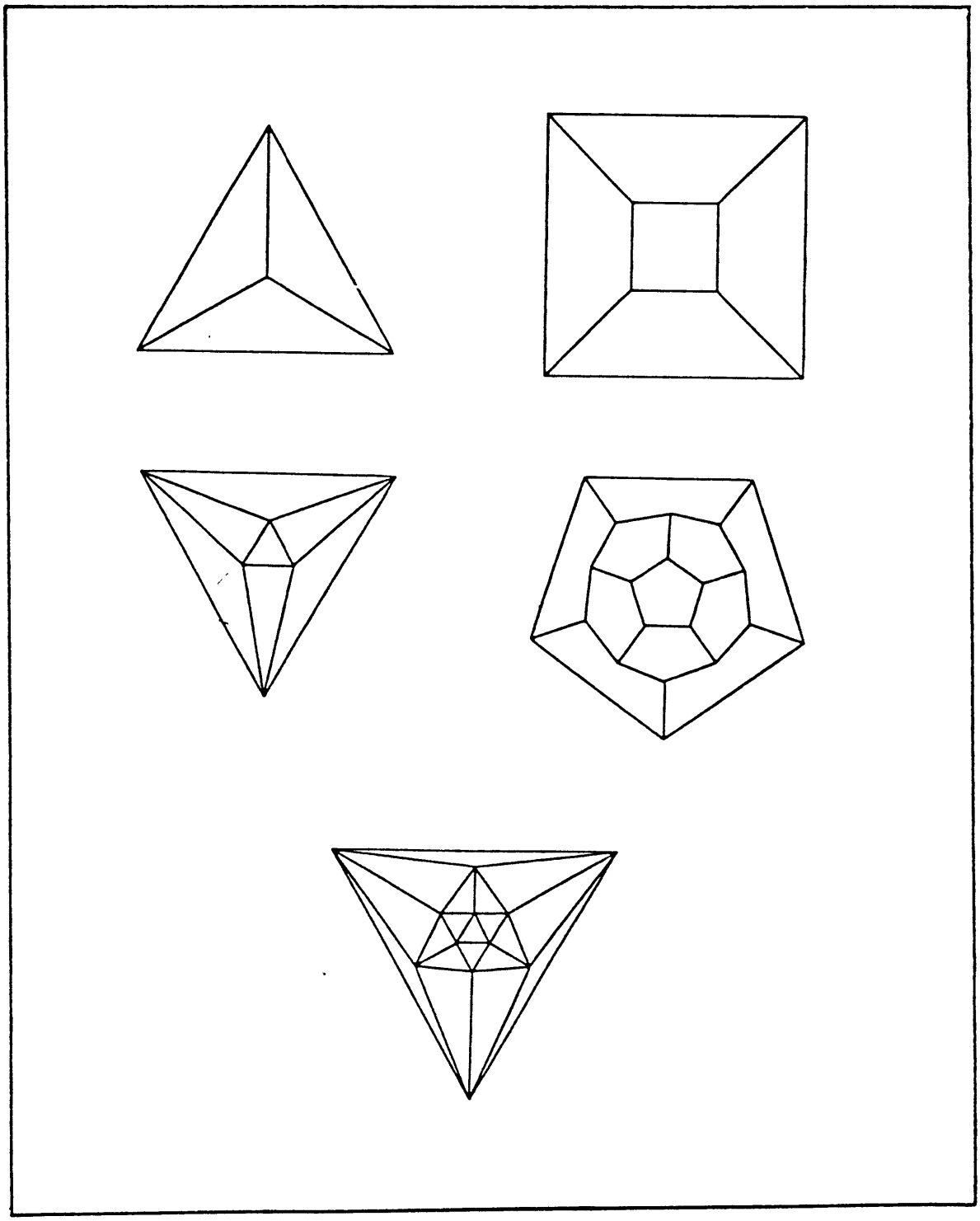


fig. 2.4 Representations of polygons in fig. 2.3 as planar graphs



bounded is finite and an infinite face has infinite area. A planar graph then has an infinite face external to all edges. Two faces are adjacent if they share a common edge, and every edge is part of exactly two contours.

Another definition, more suitable for our purpose is presented as theorem 7.

Theorem 7 (MacLane): The set of contours of the finite faces of a planar undirected graph forms a basis of the cycle subspace of the graph.

Proof: A cycle of a planar graph encircles one or more than one face. The edges of a cycle are then equal to the ring sum of the set of contours of the encircled regions. Thus any cycle or edge disjoint union of cycles can be expressed as a linear combination of the contours of the finite faces. Moreover, no contour of a finite region can be formed by a linear combination of contours of other finite faces. This is because the ring sum of the contours of two or more finite faces is a cycle (or edge disjoint union of cycles) containing those areas of the finite faces. ■

There are e - n + 1 finite faces and therefore the cycle basis has rank equals to e - n + 1 (which is also equal to the cyclomatic number). Therefore, we can use the set of contours of the finite faces in place of a fundamental set of cycles as the cycle basis and

use the method of Section 2.1 (Cycle Vector Space Methods) to generate all the elementary cycles in the graph.

To apply theorem 7 however, we need to first ensure that the graph is planar and obtain a plane representation of the graph. This problem is solved by Hoftcraft and Tarjan [16], requiring computation time bounded by a polynomial in n.

The planar algorithm starts constructing the planar graph from a planar subgraph (usually a cycle) of the given graph. Gradually, the remaining edges are added to the selected planar subgraph, such that no edges cross. The way to know this is by testing if the number of edges and vertices satisfy Euler's condition, namely, if $e > 3n - 3$, we know that the given graph is non planar. Hoftcroft and Tarjan [16] uses the depth-first-search method (see Tarjan [33]) for their algorithm. The graph is also represented as an adjacency list which incurs lesser storage and examination time. The program is written in Algol and when tested on an arbitrary graph with 900 vertices, it requires less than 12 seconds of running time.

Having an algorithm for testing and representing planarity of the graph, we now move on to introduce an algorithm that generates all the elementary circuits in a directed planar graph. Before doing so, we need to first acquaint ourselves with dual graphs.

Let D be the dual graph of a planar undirected graph G. For each face in G, let there be a corresponding vertex in D. Also, for each edge common to two adjacent faces in G, let there be a

corresponding edge which joins the two vertices corresponding to the adjacent faces, in D.

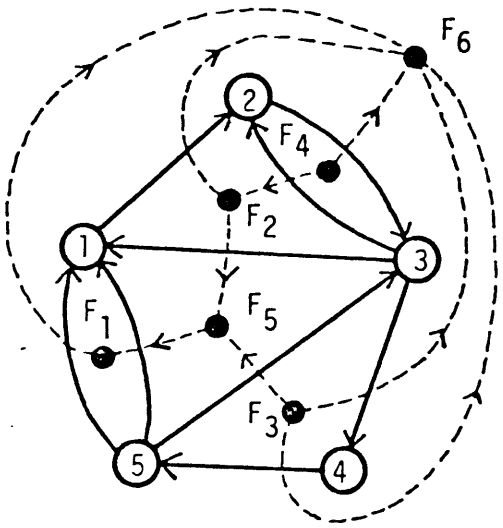
Every graph that is planar has a dual, and the dual of a dual graph is the planar graph itself (Whitney). There is a one-to-one correspondence between the number of edges in G and D. The contour of a face in G corresponds to a cutset separating the corresponding vertex from other vertices in D. But this corresponding set of cutsets in D is a basis of the cutset subspace in D. From theorem 7, we know that the set of contours of the finite faces is a basis of the cycle subspace of G. Thus, there exists a one-to-one correspondence between the vectors in the cycle subspace of a planar graph and the vectors in the cutset space of its dual and vice versa. As such, the problem of enumerating the cutsets of the dual graph is equivalent of enumerating the cycles in the primal graph.

These statements apply to undirected planar graphs. We are now ready to extend our understanding of dual graphs to generate all the elementary circuits in a directed planar graph.

Given a directed planar graph G_d, construct the dual D_d as follows; first, obtain the dual of the undirected version of the directed graph. In addition, if an edge from a vertex in the dual intersects a planar arc which is oriented in a clockwise direction around the dual vertex, then the dual edge is directed outwards from it, otherwise, it is directed inwards. Consider the example in fig. 2.5 . Note that the graph, G_d in fig. 2.5 is similar to that in Example 2,

except that we have removed the self loops.

fig. 2.5 Directed Graph G_d and its Dual D_d



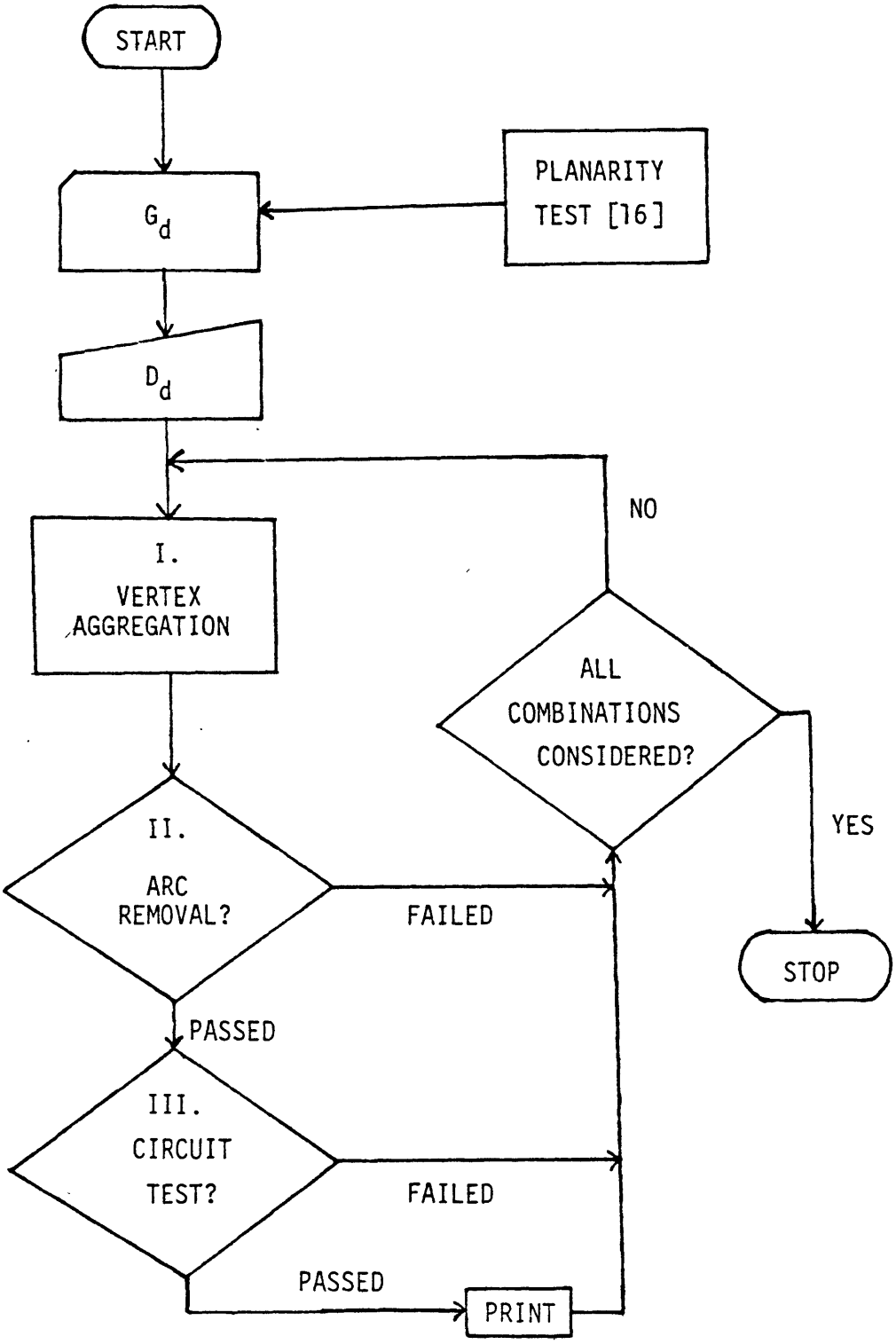
------ Dual Graph, D_d.

_____ Planar Graph, G_d.

We have labelled the faces in the planar directed graph G_d, F_1, F_2, F_3, F_4, F_5, F_6, (where F_6 is the infinite face) and the corresponding vertices in the dual graph, D_d, F_1, F_2, F_3, F_4, F_5 and F_6. In general, we shall always label faces in the planar directed graph and the vertices in the dual as $F_i = \{F_1, F_2, \ldots F_{e-n+2}\}$, where F_{e-n+2} is the infinite face.

The method for generating all evaluation circuits in a directed planar graph is presented as a flowchart in fig. 2.6 and works

fig.2.6 Flowchart on obtaing all elementary circuits in a directed planar gaph, G_d, using the dual graph, D_d.



stricly on the dual graph D_d. It consists of three main blocks:

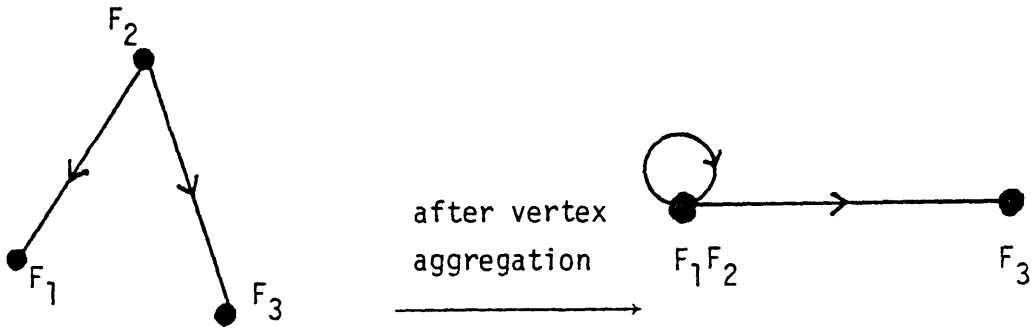
I. Vertex aggregation,

II. Arc removal, and

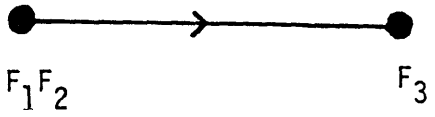
III. Circuit test.

Vertex aggregation involves taking distinct combinations of vertices in the dual that do not include vertex F_{e-n+2}. One of the ways of taking distinct combinations from the e - n + 1 vertices has been introduced by the algorithm in the previous subsection. For a graph of e - n + 1 faces, we need to consider $2^{e-n+1} - 1$ combinations.

For each combinations of x vertices, we have to ensure that we remove at least x - 1 arcs following vertex aggregation. If this condition is satisfied, then block II, "arc removal" has been accomplished. For example, consider the dual graph:



after arc removal,



Vertex aggregation and arc removal obtains a dual of the planar subgraph where those corresponding arcs are removed in the planar. Each arc so removed is common to two adjacent faces. Each vertex aggregation also forms a new dual vertex. If "arc removal" is successfully accomplished, then we proceed with circuit testing in block III.

Circuit testing involves vertex inspection. A circuit in the directed planar graph corresponds to a vertex in the dual graph with all incident arcs directed either inwards or outwards only. Note, circuit testing is only performed if "arc removal" is successful; that is, for x vertices aggregated, at least x - 1 arcs are removed.

Let us summarize our method. We start off with the dual graph, D_d, and aggregate on the vertices corresponding to the finite faces in the planar. Vertex aggregation is followed by "arc removal," which

essentially removes the arcs that are common to the corresponding faces associated with the vertices aggregated. With each vertex aggregation we obtain a new dual vertex, and providing "arc removal" is sucessful, we test to see if these combined faces (with common arcs removed) form an elementary circuit; that is, if all incident arcs are directed inwards or outwards from this new dual vertex, then we have formed a new elementary circuit.

We now use the graph in fig. 2.5 as an example to illustrate our method. The results are shown in Example 5. The example reveals the limitation of this method. For $e - n + 1 = 5$, the number of vertex aggregation that have to be considered is $2^5 - 1 = 31$. Clearly, as the number of faces increases, the number of vertex aggregation increases exponentially. Note however, that the Cycle Vector Space Method of Gibbs-Welch discussed in Section 2.1 also requires $2^{e-n+1} - 1$ combinations for comparisons in the worst case. Moreover, what we have presented here is applicable to a directed graph, even though it has to be planar. In essence, what we have really established is that the vertices (excluding F_{e-n+2}) in the dual planar graph forms a basis for finding the directed elementary circuits in the planar graph. Also, notice that, in most cases "arc removal" reduces the number of aggregation to be considered enormously. Another way of viewing "arc removal" is that the vertices that are aggregated must form connected components. The test for connected components is available (see Narsigh Deo [10]). Circuit testing via vertex inspection is done by

Example 5. Generating Elementary Circuits in a Directed Planar Graph Using the Dual

✓: Passed x: Failed NA: Not Applicable NC: Not Connected

Note: The procedure proceeds from left to right for each vertex aggregation.

Vertex Aggregation	Arc Removal	Circuit Test	Circuit
F_1	✓	x	-
F_2	✓	x	-
F_1F_2	x: F_1,F_2 NC	NA	-
F_3	✓	✓	F_3: contours of face F_3
F_1F_3	x: F_1,F_3 NC	NA	-
F_2F_3	x: F_2,F_3 NC	NA	-
$F_1F_2F_3$	x: F_1,F_2,F_3 NC	NA	-
F_4	✓	✓	F_4: contours of face F_4
F_1F_4	x: F_1,F_4 NC	NA	-
F_2F_4	✓	✓	F_2F_4: contours of F_2, F_4 excluding common arc
$F_1F_2F_4$	x: F_1,F_2,F_4 NC	NA	-
F_3F_4	x: F_3,F_4 NC	NA	-
$F_1F_3F_4$	x: F_1,F_3,F_4 NC	NA	-
$F_2F_3F_4$	x: F_2,F_3,F_4 NC	NA	-
$F_1F_2F_3F_4$	x: F_1,F_2,F_3,F_4 NC	NA	-

Vertex Aggregation	Arc Removal	Circuit Test	Circuit
F_5	✓	x	-
F_1F_5	✓	x	-
$F_1F_2F_5$	✓	x	-
F_3F_5	✓	x	-
$F_1F_3F_5$	✓	x	-
$F_2F_3F_5$	✓	x	-
$F_1F_2F_3F_5$	✓	x	-
F_4F_5	x: F_4,F_5 NC	NA	-
$F_1F_4F_5$	x: F_1,F_4,F_5 NC	NA	-
$F_2F_4F_5$	✓	x	-
$F_1F_2F_4F_5$	✓	x	-
$F_3F_5F_5$	x: F_3,F_4,F_5 NC	NA	-
$F_1F_3F_4F_5$	x: F_1,F_3,F_4,F_5	NA	-
$F_2F_3F_4F_5$	✓	✓	$F_2F_3F_4F_5$: contours of F_2,F_3,F_4,F_5 excluding common arcs
$F_1F_2F_3F_4F_5$	✓	✓	$F_1F_2F_3F_4F_5$: contours of F_1,F_2,F_3,F_4,F_5 excluding common arcs

********** End of Example 5 **********

finding whether there is one incident arc orientated in the opposite direction. The existence of one pair of opposite incident arcs in an aggregated vertex is sufficient to establish that no circuit is formed.

Everything we have discussed thus far works well provided we can easily obtain the dual from the planar graph. This remains a difficult problem since there is no algebraic representation of a face in the planar graph. As such, our ability to construct the dual is restricted to working on the planar geometric representation of the graph. Perhaps one way to facilitate this task is to keep track of when the path added in Hoftcroft and Taijan's algorithm forms a closed region. At that point we could then identify the dual vertex associated with that face.

Another possibility is to construct the dual from the planar displayed on the console of a computer terminal. We suspect that the complications would very much depend on how well we are able to represent our planar graph. In general, however, the planar graph allows us to "stretch" and "pull" the graph in order that faces are more easily identified.

It seems then that the choice of this method rests rather heavily on how easy it is to obtain the dual graph from the planar.

This concludes our review and suggestions on circuit and cycle enumeration methods.

In the next section, we compare some algorithms using the methods we have just reviewed and provide some recommendations.

Section 3

Analysis of Algorithms

In the previous section , we presented the methods for geration of cycles or circuits[10] in a graph. This section examines how one algorithm compares with another in terms of running time and storage requirements in the worst-case. Worst-case analysis is by no means indicative of the efficiency of an algorithm for an actual real-life problem. However, it does provide us with a means for comparison as well as an indication of the limitations of the algorithm. In effect, the performance of each algorithm depends strictly on the specific structure of each graph we are working with.

This section starts with ways of reducing and editing the graph without losing any cycle or circuit that belongs to the original graph. These reductions and editions help make circuit generation a more realistic and plausible choice as an approach to some problems where the problem of circuit enumeration would be enormous and intractable in every sense of the word.

Having reduced and edited the graph, we then proceed to compare the worst-case performance of all the algorithms that fall into the methods we have discussed. Each of the authors referenced has been cited previously. Worst-case analyses are done on graphs that have been reduced and edited under the rules in section 3.1.

Thus far, the connection between directed and undirected graphs

remains somewhat obscure. In the subsection following, we present a method suggested by Ardon and Malik [1] for converting an undirected graph to a directed graph but maintaining the one to one correspondence between cycles in the undirected and circuits in the directed graph. This method avoids having to enumerate each cycle twice in the corresponding directed version of the undirected graph. It also presents an option for enumerating cycles in an undirected graph using a directed circuit algorithm.

Finally, we suggest which algorithm is best in each of the methods discussed and from those, which would we recommend in general.

3.1 Graph Reduction and Edition

The problem of enumerating cycles or circuits is almost always intractable for an original graph before graph reduction and edition. The reason for this is that all the algorithms that we have presented are exponential to the size of the graph. Given a graph, it is possible to apply a set of rules to the graph in order to reduce the size of the graph in such a way that no cycles or circuits are omitted from the original graph. Although this does not guarantee that the computational problem can be reduced, it offers some improvements in most cases. Moreover, the reduced graph is not only smaller but more manageable and suitable for the methods we have discussed. In fact, all algorithms work better with the reduced graph. Furthermore, all algorithms for worst-case analysis apply to graphs that are reduced and edited.

The rules for reducing and editing graphs are as follows:

1. Self loops can be enumerated first, afterwhich they need not

be considered anymore since no other elementary circuit can contain them. In algorithms that "grow" with e, we can reduce e by the number of self-loops in the graph.

2. A set of p parallel edges or arcs can be replaced by a single edge or arc. Thereafter, whenever a circuit is found that contains this arc, p such circuits are enumerated. Again, this reduces the size of e in the graph. Furthermore, parallel arcs are not easily distinguishable in the search procedures as discussed in section 2.2.
3. Vertices of both indegree and outdegree of one can be deleted and replaced by a single arc or edge. This not only reduces the number of arcs or edges and vertices in the final graph, but also the cardinality of cycles or circuits. In algorithms where cycles or circuits of small cardinality are enumerated quickly and shortly after the start, this reduction is extremely helpful.
4. Vertices of zero indegree or zero outdegree are deleted together with their incident edges or arcs, because these cannot be part of a cycle or circuit. This process is continued until no such vertices remains. This way of pruning the graph can be visualized as chopping off the "branches" that stick out from the graph. As a result we have a more compact graph.
5. The classical "divide and conquer" rule: For a directed graph with "strong components", it is possible to decompose the problem into finding all the circuits for each "strong component."

In appendix A, we show how "strong components" can be obtained easily from a given graph. A maximal strongly connected subgraph of a graph is defined as a strong component of the graph. A subgraph is strongly connected if there is at least one directed path from every vertex to every other vertex in the subgraph. (See Christofides [7].) Thus, all circuits in a graph can be found in the strong components of the graph and no circuit exists which connects strong components. The usefulness and importance of this decompostion cannot be overlooked in enumerating elementary circuits in a graph. The strong components are subgraphs that are small enough hopefully so that even though our algorithms are exponential, we could generate all the circuits in these strong components quickly. Otherwise, having the strong components would tell us whether it would be wise to proceed with circuit enumeration if all strong components are large or, to enumerate all circuits from certain chosen strong components.

The reduction process is helpful since it only has steps of the order of (n+e).

3.2 Discussion of the Time and Space Bound of all Circuit and Cycle Enumeration Algorithms

We can now turn to a comparison of the performance of these algorithms, see table 1, where results from Matei and Deo [23] are presented.

Paths and circuits can be listed as edge, arc, or vertex sequences.

Table 1. Upper Bounds on Time and Space for Elementary Circuits and Cycle Generating Algorithms

Algorithm Used	Time Bound	Space Bound	Method Used
Hsu and Honkanen	$n \cdot 2^{2\mu}$	$e \cdot 2^{\mu}$	Cycle Vector Space
Mateti and Deo	$\mu^2 \cdot 2^{2\mu}$	μ^2	"
Maxwell and Reed	$n \cdot 2^{2\mu}$	$e \cdot 2^{\mu}$	"
Rao and Murti	$e \cdot \mu^2 \cdot 2^{\mu}$	$\mu \cdot n$	"
Welch-Gibbs	$n \cdot 2^{2\mu}$	$e \cdot 2^{\mu}$	"
Berztiss	$n(\text{const.})^n$	$n + e$	Search and Backtrack
Johnson	$(n + e)c$	$n + e$	"
Ehrenfeucht	$n^3 + n \cdot e \cdot c$	$n + e$	"
Read and Tarjan	$(n + e)c$	$n + e$	"
Szwarcfiter + Lauer	$(n + e)c$	$n + e$	"
Tarjan	$n \cdot e \cdot c$	$n + e$	"
Tiernan	$n(\text{const.})^n$	$n + e$	"
Weinblatt	$n(\text{const.})^n$	$n \cdot c$	"
Ardon and Malik	$n(\text{const.})^n$	n^2	Connection Matrix
Danielson	$n(\text{const.})^n$	$n(\text{const.})^n$	"
Kamae	-	$n(\text{const.})^n$	"
Ponstein	-	$n(\text{const.})^n$	"
Yau	-	$n(\text{const.})^n$	"
Cartwright + Gleason	$n(\text{const.})^n$	$n(\text{const.})^n$	Directed Graph Transformation

Note: n = number of vertices
e = number of edges or arcs
$\mu = \nu(G)$ = cyclomatic number = e - n+ 1
c = number of elementary circuits.

Source: Mateti and Deo [23]

Each vertex or arc (edge) counts as a unit of input. Space bounds or bounds on computer storage required are measure as units of input. Any operation on a vertex or edge (arc) counts as one time unit. For instance, in the Search and Backtrack Methods, whenever a path is extended to a vertex, we count that as one time unit. Time bounds for the algorithms are given in terms of time units.

The results given in table 1 are worst-case bounds. The worst-case graphs for different algorithms are in general different. It is also inportant to note that the effectiveness of an algorithm is very much graph dependent. For instance, it is easier to apply a cycle enumeration algorithm to an undirected graph instead of a circuit enumeration algorithm. In addition, the choice of the algorithm depends also on the problem we are interested in solving. An understanding of our input, or problem is almost as important as choosing the algorithm itself. Some algorithms that perform well for graphs of reasonable size might be very inefficient for graphs that are small.

From table 1, note that the Search and Backtrack Methods require the least storage. Other methods like the Vecter Space, Connection Matrix or Directed Graph Transformation method store the input as an adjacency matrix or slight variation thereof, instead of arc listings, or an adjacency list. In addition to storing the information in a huge memory, the examinations cannot stop without going through each element of the matrix. In this sense, a huge storage incurs a penalty on the running time as well

The Search and Backtrack Methods, though not theoretically as elegant

as the Cycle Vector Space or the Connection Matrix Methods, are very suitable for computer implementation. Most of the algorithms in these methods, for example Tarjan [32] and Johnson [18], uses what is known as the "Depth-First-Search" Method, see Tarjan [33]. Depth-First-Search is a procedure to search exhaustively without traversing any path twice.

Even though in general inferior, the Connection Matrix Method could be used as effectively (and in some cases more so) to enumerate special circuits like the Hamiltonian circuits, see Ardon and Malik [1], Danielson [8] and Yau [37]. The main problem with the connection matrix method is that it remains difficult to sort out the circuits in the diagonal even if we are able to eliminate nonsimple paths. So, as long as sorting remains difficult it is unlikely that the Connection Matrix Methods would gain as much attention as the Search and Backtrack Methods.

Although the algorithm of Cartwright and Gleason [3] allows for much flexibility in the way we enumerate circuits from line graphs, (see Section 3.4) it suffers almost the same problems as faced by Connection Matrix Method. Because of this, the Directed Graph Transformation Method remains relatively unexplored; and their space and time bound exhibits similarity with the Connection Matrix Method.

The Cycle Vector Space Methods consist actually of two distinct phases. Phase one generates the fundamental set of cycles, and phase two the elementary cycles of the graph. Different algorithms for generating fundamental cycles differ in both storage and running time capabilities. In Section 3.4 we recommend Paton [25]; however, this need not be. As a result, the storage requirements showed slight

fluctuations in the Cycle Vector Space Methods.

Before proceeding to compare algorithms individually, and making recommendations, we would like to point out that though it is not possible to use a Cycle Vector Space algorithm on a directed graph, without being terribly inefficient, the transformation of an undirected graph to a directed graph by Ardon and Malik [1] provides another option for enumerating cycles, should we remain unsatisfied with the Cycle Vector Space Methods. The following subsection discusses how this can be done cleverly.

3.3. Conversion of an Undirected Graph to a Directed Graph for Circuit Enumeration

Having observed in Section 3.2 that the Search and Backtrack Methods are generally faster than the Vector Space Methods (or for whatever reasons), we might be interested in enumerating cycles using a circuit enumeration algorithm. To do so, we have first to convert the undirected graph to a directed graph. Representing each edge as two arcs with opposing directions is inefficient since an edge in the undirected graph corresponds to a 2-circuit in the equivalent directed graph. We have to do better than that.

A method by Ardon and Malik [1] has been proposed so that we maintain a one to one correspondence between cycles in the undirected graph G(V,E) and circuits in the equivalent directed graph G'(V',E'). Consider an undirected graph with no parallel edges or self loops. An edge e_{ij} connects vertices i to j. We denote this edge $e_{ij} \in E$ as $(e'_{ji}$ and $e'_{ij}) \in E'$. For a given cycle in G, there is a corresponding pair of directed circuits

in G'. In other words, if there are e edges and c cycles in G, there would be e + 2c circuits in G'. We could improve on this. If we remove an arc e'_{ij} from G', all circuits in G' containing arc e'_{ij} would be discarded. However, there now exists a one-to-one correspondence between the cycles in G containing edge $e_{ij} = e_{ji}$ and circuits in G' containing e'_{ji}. That is, by deleting e'_{ij}, all cycles in G containing e_{ji} or e_{ij} are equivalent to circuits in G' with e'_{ji}. Once all cycles in G containing e_{ij} or e_{ji} are found, the arc e'_{ji} is removed from G' to form a new subgraph. The remaining cycles in G are obtained from this subgraph by the same procedure.

By repeating the procedure, a subset of circuits are found which corresponds to all the cycles in the undirected graph. Since e is finite, the number of subgraphs considered is also finite.

As an illustration, suppose we are interested in generating elementary cycles in an undirected graph using the Search and Backtrack Method discussed earlier. First, we transform G to G', then pick a vertex i, and remove an arc incident to it, e'_{ji}. Proceed by building a path from i one arc at a time starting with e'_{ij}. Test for circuits by Tiernan's [34] algorithm outlined in ALGORITHM 2, until we backtrack to vertex i. All cycles that contain edge e_{ij} would have been enumerated. Remove arc e'_{ij} and proceed as before from vertex j. When no arcs remain in G', all cycles in G would be enumerated. Since Tiernan's [34] algorithm finds circuits as a vertex sequence, all cycles are then represented as vertex sequence as well.

The use of any Search and Backtrack Methods represent only one way of enumerating cycles in G from circuits in G'. However, we could also

apply the converted graph to the Connection Matrix Methods. This is done by Ardon and Malik [1]. In fact, we could use any method for enumerating circuits on the converted graph.

In the subsection that follows, we shall recommend which algorithm to use for circuit enumeration and the rationale for the selection.

3.4 Recommendations

The selection of an algorithm to use for circuit enumeration should be approached with much discretion. In general, the number of cycles or circuits are enormous and happily, for most practical purposes, we are only interested in generating cycles or circuits with particular properties. For example, in a complete directed graph, where every vertex is connected by an arc, there are n(n-1) arcs and the number of circuits of length i is equal to $\binom{n}{i}(i-1)!$. Thus, the total number of circuits is $\sum_{i=2}^{n} \binom{n}{i}(i-1)!$. But $\sum_{i=2}^{n} \binom{n}{i}(i-1)! \geq 2^n - n - 1$; this means that the number of circuits grows with the exponential 2^n for such graphs.

Like most problems then, it is essential to first understand and "visualize" the nature of the problem. Though this is not always possible, it would lead to substantial savings if we have an idea of what the original graph looks like, and what we want to do with it (i.e., do we need to enumerate cycles or circuits?) Indeed, the selection of an algorithm rests on such criteria alone. For instance if we have a complete graph, to enumerate circuits in such a graph using any algorithm would be impossible for large graphs. Also, we might be able to focus our attention in a subgraph of the original graph and identify all circuits

in that subgraph without incurring huge penalty to the original problem.

First, we note that an algorithm which is superior theoretically is not necessarily superior in practice. For instance, the simplex method is still better for solving linear programming problems than Khachiyan's algorithm even though the latter is polynomially bounded and the former is not.

Also, suppose two algorithms α and α_p solves a problem Q, where α_p is polynomially bounded and α is not. Then there is some family of instances $\{Q_n\}$ of Q such that the running time of α on $\{Q_n\}$ increases faster than the polynomial function of n, while the running time of α_p on $\{Q_n\}$ is bounded by some polynomial function f(n). For "large enough" values of n, α_p is guaranteed to run faster on Qn than α, and as n increases, the discrepancy increases rapidly. This result is known as an asymptotic result. But, how large is "large enough?" In lieu of this, it is possible that the polynomial algorithm α might be preferable to α_p for all instances of Q.

Another point we like to make here is that polynomial boundedness is pathologically contrived such that they represent the most perverse problem instances in order to measure an algorithm's performance. Besides, how likely are we in practice to encounter problem instances like those in $\{Q_n\}$ that causes α to behave badly.[11]

Even so, since most polynomial-time algorithm for problems of interest to Operations Research are both efficient in theory and practice, a comparison on that basis would be beneficial.

Algorithms are compared under three criterias: worst-case performance,

clarity and programmability.

For an undirected graph, the author recommends the Cycle Vector Space Methods and in particular, the Welch-Gibbs[12] algorithm (with Paton's [25] algorithm for finding the fundamental cycle set). Paton's [25] algorithm has been shown to out perform other algorithms for finding the fundamental cycle set. Although the Welch-Gibbs algorithm does not perform any better or worse than the others in the same category, it is clearly expounded and easily programmed. It has been programmed in Fortran and checked for approximately 100 graphs of at most 25 vertices each. On the CDC 6500, it took about two minutes to generate all cycles of 25 12-point cubic graphs with the input in the form of a vertex adjacency matrix for each graph.[13]

For a directed graph, Johnson's algorithm is recommended. Though there are three other algorithms that are polynomially bounded (if the number of circuits is polynomial), Johnson's [18] algorithm has been shown to converge asymptotically fastest. The notion of asymptotic convergence has been discussed previously. The program is written in Algol and the running time for a graph of 80 vertices and 240 circuits takes 4.46 seconds on IBM 370/168. The method of Johnson builds upon the algorithm of Tiernan discussed in Section 2.2, but exploits the added features and capabilities of the computer.

Since the search procedures are not all that complicated, one suggestion would be to write the program for the algorithm in Assembler language, which would improve on the running time. (The same suggestion applies to

Welch-Gibbs algorithm.)

Though the Connection Matrix Methods incur huge storage requirements and examination time (since each element must be scanned), it is theoretically elegant and for a graph that is not sparse, it might be easier to use since we are dealing only and strictly with matrix manipulation in this method. For this category, Ardon and Malik [1] is recommended. No results on computations are available, but the treatment of circuit generation is not as elaborate as the others. Kamae's method for instance requires that we keep every matrix until the algorithm stops. The number of matrices fortunately does not depend on the size of the graph.

We do not recommend Cartwright and Gleason [3] unless a better way of extracting circuits from line graph is available.

One important point arises from the previous discussion. The algorithms listed in table 1 seem to perform, in general better the younger they were. This leads us to believe that theoretically, the problems with circuit enumeration have reached a mature level but computationally there remains room for improvement. The improvement in computer technology has enhanced our ability to tackle these problems more efficiently and indeed would alter the preference with which we select our algorithm.

Now that we have recommended one algorithm for each method, the question remains as to, "If I have to use one algorithm to enumerate cycles or circuits which should I use?"

First determine whether the graph is directed or undirected. If

directed, we recommend Johnson [18], otherwise Welch-Gibbs [14], with Paton [25] for finding the fundamental cycle set. The reason should be obvious from previous discussion. For generating all simple cycles or circuits we recommend our algorithm. For a planar directed graph, we still recommend Johnson instead of our algorithm since it remains difficult to obtain the dual from the primal.

Let us now summarize our approach to the problem of enumerating all elementary circuit or cycles in the graph:

The algorithms to solve for all elementary circuits or cycles in a graph should be used only as a last resort for solving problems that requires these circuits or cycles as feasible solutions since it is an expensive and slow process. If there are no other options, then we should first try to understand the structure of the problem in order to better visualize the graph we are working with. In other words, we should decide whether we could either reduce or reformulate the problem, or better still could solving the problem on a partial graph provide us with meaninful results?

If the graph we are working on is directed, use Johnson's algorithm, otherwise, use Welch-Gibbs (with Paton [25] for finding the fundamental cycle set).

The enumeration of all elementary cycles or circuits remains a costly and difficult problem. It would be better if we could concern ourselves with generating cycle or circuits with particular properties. This is the topic of the next section which is more relevant from a practical viewpoint.

Section 4

Applications of Circuit Enumeration

We now take a look not at the very expensive and time-consuming task of enumerating all the circuits or cycles, but at more restricted problems which find a particular subset of circuits in the graph and suggest some applications pertaining to these classes of circuits. In Section 2.5, we showed how all simple circuits or cycles could be obtained from the set of elementary circuits. It is apparent that any classification of elementary circuits with particular properties could be extended to simple circuits as well. The same applies for an undirected graph. For convenience, we shall work with elementary circuits.

4.1. Elementary Circuits with Particular Properties

All classifications must have properties of vertices, arcs, circuits or cost. (If in addition we associate a cost with each vertex or arc.) An elementary circuit having n vertices must have n arcs. Thus, finding circuits that pass through n arcs is equivalent to finding circuits passing through n vertices.

The restricted problems we have identified are:

P1. Find all elementary circuits that pass through k specific vertices.

P2. Find all elementary circuits that pass through k specific arcs.

P3. Find all elementary circuits that consist of p (or less than p) arcs (vertices).

P4. Find all elementary circuits less than a given fixed cost (with costs associated with arcs and/or vertices).

P5. Find a set of q disjoint elementary circuits which passes through a set of k vertices.

P6. Elementary circuits that are cheaper than at least r other circuits.

These six problems we have identified are mutually exclusive. Many more problems would be formulated by considering combinations of these six problems, P1 to P6. For example, if we consider P3 and P4, we would define the problem of finding elementary circuits that consist of p (or less than p) arcs (vertices) and less than a given fixed cost. Furthermore, some actual practical problems are easily associated with one or more restricted problems. In P6, if we set $r = 1$, and solve P6 repeatedly, we would get the optimal or least cost circuit in the graph. Remember that r could be set arbitrarily. P3 and P6 could define the travelling salesman problems, and finding the Hamiltonian circuit would be a special case of P3 for $p = n$. Another example is to find the optimal circuit in a doubly weighted graph

(i.e. two costs are associated with each arc), see Dantzig et al.,[9] and Lawler [21].

In case of simple circuits, we might be interested to identify a simple circuit that consists of elementary circuits of cardinality ℓ, with a common vertex.

Having proposed some restricted problems which find a particular subset of circuits in the graph, we move on next to see how some practical problems could fit into our classifications.

4.2. Some Suggested Applications

For P1, we might be interested in locating a distribution depot for serving a set of k cities (where the demand for a certain good/service is higher) such that all cities are served by one cyclic route. The assumption is that the demand is known and the cities are connected. P1 provides us with all possible circuits (but not more than we need) and choices of the location of the depot. Since the cities are connected, we could partition the network into strong components, where the set of k cities lies in one of these. We enumerate circuits contained in this strong component. If no such circuit exists, then the method would inform us. Otherwise, it would identify all such circuits.

In addition, we could extend P1 to include cyclic production scheduling and job scheduling. We denote the beginning of a production run by a vertex, where an arc connects the job, represented

as a vertex, that must be completed or required before another is undertaken. We would then like to find all circuits that passes through a fixed vertex (the beginning of the production process or job assignment). Tiernan's [34] algorithm solves this problem, but we need only to restrict ourselves to the first iteration of the algorithms; namely, generating circuits that passes through vertex 1 where vertex 1 is the fixed starting vertex. (See Section 2.2.)

For P2, we might select the set of arcs with least penalty (i.e. congestion, tolls or length). Our problem would be to dispath vehicles from a depot, through these streets, such that they return to the depot upon completion of the job. Note that we have imposed a somewhat stringent condition that the vehicle should return to the depot. To solve P2, we transform P2 to P1 by finding the "line graph" associated with the original graph and represent the k specific arcs as k specific vertices, then solve it as P1.

P3 could be solved more efficiently using the Connection Matrix Method or Directed Graph Transformation Method, since it generates all circuits of a given cardinality simultaneously. A simple scheduling problem might insist that we do not visit more than p cities at a time. For example, we might be interested in scheduling buses so that they do not have more than p stops. Other constraints might be included like the total trip length but this will be discussed later. Another interesting problem involves finding Hamiltonian circuits. This is discussed in detail in Appendix B (also included are Eulerian circuits).

An interesting extension to P3 is P4, where the total cost of a circuit is constrained to be less than a given time, or total trip length or total cost of production. If we assign weights to all arcs in the graph where weights can be negative (for example, if cost is a positive factor, then profit would be negative and vice-versa), the problem becomes, given a set of schedules or production strategies, find the schedules or strategies which do not exceed a certain time bound or do not exceed a certain variable cost. The algorithm for solving this P4 where arc costs are non negative is given in ALGORITHM 5, which is a modification of Tiernan's algorithm. We denote the fixed cost as Q. If some arc costs are negative, we suggest two approaches depending on the number of negative cost arcs:

A1. Divide the set of vertices into two disjoint sets. One set is the vertices which have negative cost arcs incident to them. Let this set contain n' elements, and the other set of vertices with non negative cost arcs incident contain n" elements. Note that n = n' + n". Label the vertices from the first set 1, 2,..., n' and the vertices in the latter set n' + 1, n' + 2,..., n. Solve the first n' set of vertices using Tiernan's algorithm to obtain all the circuits. Note also that we need to use Tiernan's algorithm only until the vertex n' is updated or incremented in B6.

(see ALGORITHM 2). No circuits formed after that would contain vertices 1 to n' and its incident arcs (i.e. no negative cost arcs would be considered again). Discard any circuits that were found which exceeds cost Q. Using the modified algorithm in Algorithm 5, we continue with finding circuits which fall within the allowable cost. Observe that since the subgraph considered in the latter problem has no negative cost arcs, and once the path cost exceeds the fixed cost, we need not proceed further to build circuit. This approach is suitable if we have a small set of negative cost arcs, i.e., if n' is small.

A2. If most of the arc costs are negative, we propose a second approach. First, we find the arc with the least cost, say $-c^*_{ij}$. We add an additional cost c^*_{ij} to all arcs. Then whenever an arc is extended in the path, we test to see if the total cost of the path is less than $Q + mc^*_{ij}$ where m is the cardinality of the path (or circuit). The modification to Tiernan's algorithm has to include the above cost test and an array that keeps track of m, the cardinality of the path (i.e. everytime we backtrack, $m \leftarrow m - 1$). In this case though, we enumerate only circuits that fall below cost Q.

One example of a problem with negative cost arcs is the scheduling of flights where some flights are profitable, i.e. with negative cost. This happens whenever there is a high demand from point A to B but not

ALGORITHM 5: Modified Tiernan's Algorithm - An algorithm for enumerating all elementary circuits that does not exceed cost Q. (All edge cost are assumed to be positive.)

B1. Initialize

Read N, G, and C

$P \leftarrow 0$

$H \leftarrow 0$

$k \leftarrow 1$

$P(1) \leftarrow 1$

$T \leftarrow 0$

B2. Path extension

Search $G(P(k),j)$ for $j = 1,2,\ldots N$ such that the following four conditions are satisfied:

1. $G(P(k),j) > P(1)$
2. $G(P(k),j) \notin P$
3. $G(P(k),j) \notin H(P(k),m), \quad m = 1,2,\ldots N$
4. $T + C(P(k),j) < Q$

If this j is found, extend the path,

$T \leftarrow T + C(P(k),j)$

$k \leftarrow k + 1$

$P(k) \leftarrow G(P(k - 1),j)$

go to B2.

If no j meets the above conditions, the path cannot be extended.

B3. Circuit confirmation

If $P(1) \notin G(P(k,j), \quad j = 1,2,\ldots N$ or $D = T + C(P(k),P(\ )) > Q$,

then no circuit has been formed,

go to B4.

Otherwise a circuit is reported,

Print P, D.

B4. Vertex closure

If $k = 1$, then all of the circuits containing vertex $P(1)$ has been considered,

go to B5.

Otherwise,

$H(P(k),m) \leftarrow 0, \quad m = 1,2,\ldots N$

For m such that $H(P(k) - 1),m)$ is the leftmost zero in the $P(k - 1)$ - the row of H,

$H(P(k - 1),m) \leftarrow P(k)$

$P(k) \leftarrow 0$

$T \leftarrow T - 1$

$k \leftarrow k - 1$

go to B2.

B5. Advance initial vertex

If P(1) = N then

go to B6.

otherwise,

P(1) ← P(1) + 1

k ← 1

H ← 0

go to B2.

B6. Terminate

End of Algorithm 5

from B to A. We would like to schedule flight routings that fall within a certain operating cost. Perhaps this example is not complete since the requirement that an aircraft returns to its homebase is not as stringent. In the hub and spoke network structure though, this requirement becomes more reasonable.

Instead of finding a single circuit that passes through k vertices, P5 provides us with q disjoint elementary circuits, which passes through k vertices. If we have q vehicles of different capacities and/or specifications, we could dispatch each vehicle to serve each circuit. This is a deviation from finding the single least cost elementary circuit in the graph (TSP). In fact, it might be more efficient and meaningful to serve several independent disjoint tours instead of one single tour which could be too large (in terms of cardinality). Besides, we might have more than one vehicle (or server) at our disposal. We could easily identify disjoint circuits since they will never share any vertex in common. In addition, we could find one circuit that contains all k vertices.

Finally, P6 provides an order of the cost of each circuit. If we let r = 1, and perform P6 repeatedly, we get an ordering of circuits from most expensive to cheapest. This problem is of interest to us in the same way the k-shortest path is for shortest path problems. It allows us to settle for next, or next to next, etc., best circuit and know how far away from the optimal we are. The reason for selecting the second cheapest, instead of the cheapest circuit is often a

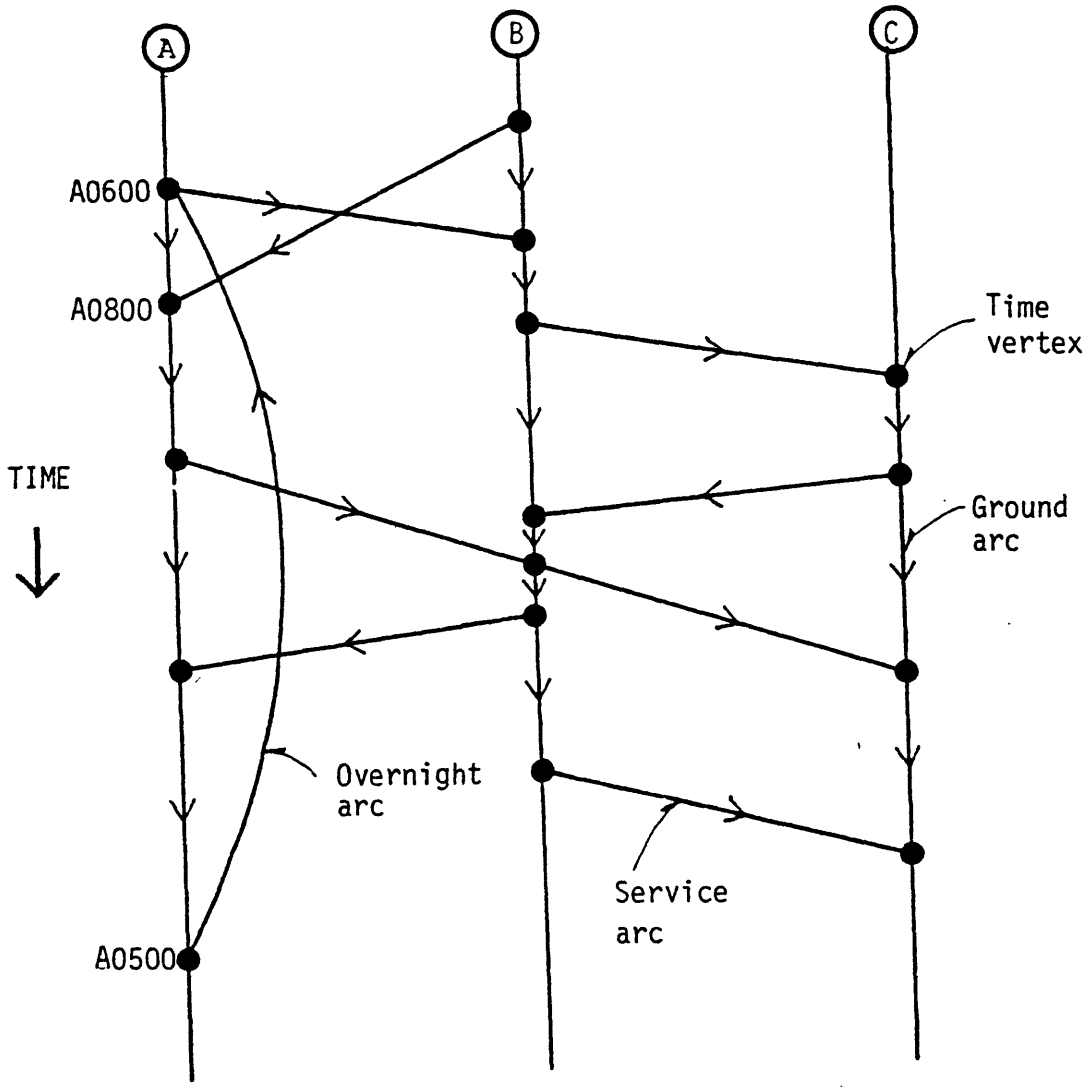
judgemental one. Whereas the k-shortest path problem is solved, the corresponding problem for circuits has not been attempted.

Combining restricted problems also define interesting areas of applications.

Let us consider a simple problem which results from combining P1 and P5. Suppose we are given an airline schedule map, and we try to determine how to schedule a crew such that they return to have base after a certain time period not exceeding Q hours. This problem is analogous to scheduling of aircraft for maintenance at home base within a time period. One problem then is to find all circuits that fall within this constraint. But first let us understand what a schedule map is.

A particular schedule map is shown in fig. 4.1. The vertex corresponding to the vertical axis are time vertices at a given city (represented as vertical axis). The arcs connecting cities are called service arcs. Ground arcs connect one time instant at a given city to another time instant at the same city. Observe too, that in our particular schedule map, only the home base, which can be any city, has an overnight arc. For purpose of illustration, let A be the home base. If we are interested in finding circuits that includes the home base and that do not exceed Q hours, then we should extend the schedule map such that the vertical axis of the home base equals Q hours. We then use the modified Tiernan's algorithm (Algorithm 5) to generate all circuits that fall within this time limit Q. If we denote our home base as vertex 1, then we only need to find circuits that pass through vertex 1 only (i.e. no "advance vertex" is necessary).

fig. 4.1 Schedule Map



Source : Simpson [30]

In the case where there are restrictions on routes or precedence relationship for visiting one (or some) vertex before visiting another, a graph transformation involving vertex splitting should be done prior to carrying out the algorithm. Vertex cost could also be represented on the arc between the split vertex.

The enumeration of all circuit and cycles have also been used to solve subproblems. For example, the airline crew scheduling problem uses the set of rotations as inputs. The set of rotations is obtained from the set of elementary cycles. Another example is the subtours in a tanker routing problem where the set of circuits represent the feasible set for selecting the cheapest tour.[14]

Section 5

Summary

We have shown in section 2 how to enumerate all elementary cycles or circuits from a graph. We have also shown how to obtain all simple cycles or circuits from a given set of elementary cycles or circuits. In addition, an approach using the dual graph to find all elementary circuits in a directed planar graph is introduced in section 2.6.

Next, we compared the running time and storage requirements of most cycle and circuit enumeration algorithms in table 1. We then proceeded to make recommendations in section 3.4 based on these computational results.

However, we have seen that even though, theoretically, we can solve the circuit enumeration problem, it remains as one of those problems which requires exponential time to solve. For instance, if we are trying to find all the elementary cycles in the graph, given that the graph contains 60 fundamental cycles; and that each operation requires 1 micro second to compute, then to exhaustively consider all possible combinations of fundamental cycles would require $2^{60} - 60 - 1$ micro seconds, which is about 366 centuries! Other algorithms are polynomially bounded, but only if we know a priori that the number of circuits is bounded. This is comforting, but knowing the number of

circuits in a graph is not always possible. No one seems to have an algorithm to do this.

In any case, if we need to find all elementary circuits or cycles using any of the algorithms referred to in section 2, then we recommend that the graph is first edited and reduced following the rules listed in section 3.1. This would save us much computational time, or indicate that the problem is too large for us to even try solving.

Where do we go from here?

We have isolated some circuit problems in Section 4.1 and suggested applications in Section 4.2. Though these problems could be solved using existing algorithms, and modifications, there are still room for improvements. We recommend developing algorithms for problems listed in Sections 4.1 and 4.2 and try to improve on computational efficiencies.

In addition, we might also try to find a subset of all cycles or circuits (e.g. the fundamental set of cycles) in polynomial or pseudo-polynomial time, then later to use this set as a surrogate for all the cycles or circuits in the graph for further analysis if needed. (Note that the subset of cycles provided by the fundamental set might not be satisfactory for further analysis.) One other example of finding subsets of all circuits would be to find all the circuits from selected strong components that are smaller than a given size.

Finally, it would be interesting to be able to simply count the number of elementary cycles or circuits in a graph efficiently. Having this knowledge would allow us to use Johnson's [18] algorithm, which

is polynomial to the number of circuits in the graph with more confidence.

Though application of most algorithms are constrained by the size of the problem, the ability to handle bigger and bigger problems expands with the continuous improvement in the capabilities of the computer and software as time goes on. Fortunately, problems do not get larger and larger with time. In this sense, we continue to move towards being able to solve practical problems which require cycle or circuit enumeration.

Footnotes

[1]For the proof, the reader is referred to C.L. Liu [22].

[2]For a given spanning tree, the set of the n - 1 cutsets corresponding to the n - 1 branches of the spanning tree is called the set of fundamental cutsets relative to the spanning tree.

[3]A non-simple path uses an arc more than once. We have referred to them as infinite paths in our definitions in Section 1.

[4]$(C_1)_{ij}$ denotes the ij element of matrix C_1, i.e. brackets identify the elements of the matrix.

[5]A vertex sequence P is a flower if there exists a vertex j in P such that the subsequence of P consisting of all vertices and arcs up to j is a path, the subsequence of P consisting of all vertices and arcs behind j, including j is a circuit containing more than one arc, and that these subsequences have no vertices except j in common. The vertex j is known as the "joint" vertex.

[6]Of course, if there are alot more circuits of higher cardinality, convergence might still be slow, but in this case, the other methods would not get around this problem any easier either.

[7]For a listing of the ways of graph representation, see Narsingh Deo [10], pages 270 to 273.

[8]Note: $V(P + S_j) = V(P) + V(S_j)$
In particular, $V(S_1 + S_3) = V(S_1) + V(S_3)$.

[9]For more definitions on planar graphs, see Harary [B5]. We present definitions here that serve the discussion that follows. There are several equivalent definitions of planar graphs.

[10]We refer to elemetary cycles and circuits in this section unless otherwise specified.

[11]For further discussion on computational complexity, refer to [B7].

[12]Welch-Gibbs algorithm actually refers to Gibb's algorithm. But Gibb's algorithm is merely a modification of Welch's.

[13]Gibb uses Gotlieb and Corneil's algorithm for generating the cycle set and we can expect the running time to improve as well since capabilities in computers have improved also.

[14]The tanker routing problem have also been solved as a shortest route problem. See Dantzig et al. [9] and Christofides [7].

Appendix A

Method for Finding Strong Components (Christofides [7])

Let $R = [r_{ij}]$ be a reachability vertex where,

$$r_{ij} = \begin{cases} 1, & \text{if vertex } x_j \text{ is reachable from vertex } x_i \\ 0, & \text{otherwise} \end{cases}$$

and $Q = [q_{ij}]$ be a reaching matrix where,

$$q_{ij} = \begin{cases} 1, & \text{if vertex } x_j \text{ can reach vertex } x_i \\ 0, & \text{otherwise} \end{cases}$$

Using our usual definition of $\Gamma(\cdot)$ and $\Gamma^{-1}(\cdot)$, we define $R(x_i)$ as the set of vertices that can be reached from a given vertex x_i.

Specifically

$$R(x_i) = \{x_i\} \cup \Gamma(x_i) \cup \Gamma^2(x_i) \cup \ldots \Gamma^p(x_i)$$

where $\Gamma^p(x_i)$ is the set of vertices which are reachable from vertex x_i along a path with cardinality P.

We define $Q(x_i)$ similarly as the set of vertices which can reach vertex x_i. Therefore,

$$Q(x_i) = \{x_i\} \cup \Gamma^{-1}(x_i) \cup \Gamma^{-2}(x_i) \cup \ldots \cup \Gamma^{-p}(x_i)$$

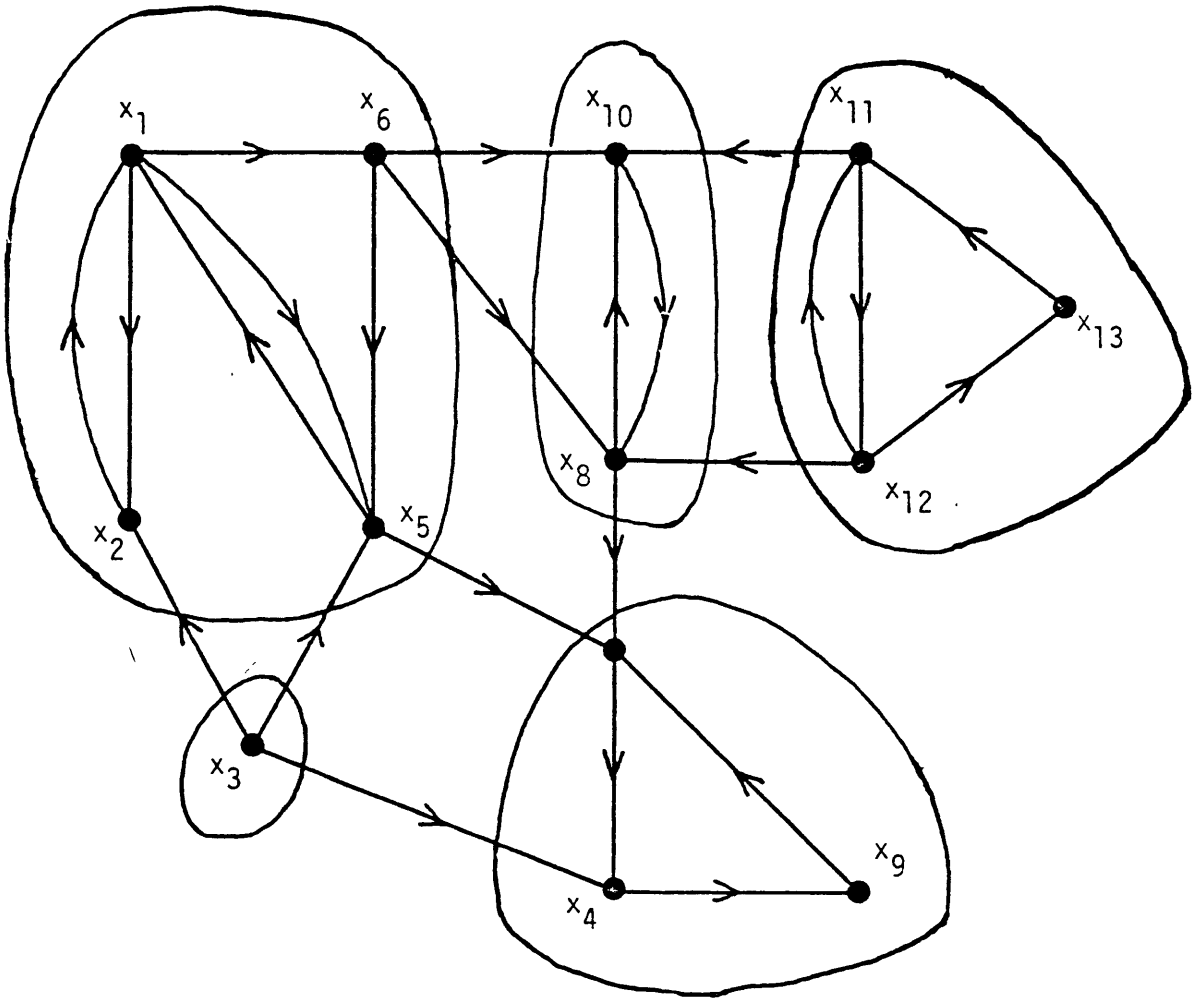
where $\Gamma^{-2}(x_i) = \Gamma^{-1}(\Gamma^{-1}(x_i))$ etc.

Observe that $Q = R^t$, the transpose of the reachability matrix. (Column x_i of matrix Q can be found by setting $q_{ij} = 1$ if $x_j \in Q(x_i)$, and $q_{ij} = 0$ otherwise.)

The set $R(x_i) \cap Q(x_j)$ is the set of vertices which are on at least one path going from x_i to x_j. Thus the set $R(x_i) \cap Q(x_i)$ is that set of vertices that can reach and can be reached from x_i and from each other. Thus $R(x_i) \cap Q(x_i)$ defines the vertices of the unique strong component of G containing vertex x_i. These vertices are then removed from G to obtain the subgraph $G' = \langle G - R(x_i) \cap Q(x_i) \rangle$. We find another strong component containing $x_j \in G - R(x_i) \cap Q(x_i)$ in the same way. This process is continued until all vertices in the graph belongs to one component. See example in fig. A.1.

Alternatively, we could find the strong components of the graph in fig. A.1 by taking the element by element multiplication of R and Q, i.e. $R \otimes Q$. The row x_i of the matrix $R \otimes Q$ contains 1 in those columns of x_j only if x_i and x_j are mutually reachable and 0 otherwise. Thus, two vertices are in the same strong component if and only if their corresponding rows (or columns) are identical. The vertices whose

fig. A.1



$$R(x_1) = \{x_1,x_2,x_4,x_5,x_6,x_7,x_8,x_9,x_{10}\} \cap Q(x_1) = \{x_1,x_2,x_3,x_5,x_6\}$$

Therefore the strong component containing x_1 is $R(x_1) \cap Q(x_1) = \{x_1,x_2,x_5,x_6\}$. Similarly, the strong components containing vertices x_7 is $\{x_4,x_7,x_9\}$, x_8 is $\{x_8,x_{10}\}$, x_{11} is $\{x_{11},x_{12},x_{13}\}$ and x_3 is $\{x_3\}$.

corresponding rows contain an element of 1 under column x_j, then forms the vertex set of the strong componentt containing x_j. By rearranging rows and columns, we could obtain a block diagonal form of the matrix $R \otimes Q$ such that each block corresponds to a strong component of G containing only 1's, see fig. A.2.

$R \otimes Q =$

	x_1 x_2 x_5 x_6	x_8 x_{10}	x_4 x_7 x_9	x_{11} x_{12} x_{13}	x_3
x_1 x_2 x_5 x_6	1	0	0	0	0
x_8 x_{10}	0	1	0	0	0
x_4 x_7 x_9	0	0	1	0	0
x_{11} x_{12} x_{13}	0	0	0	1	0
x_3	0	0	0	0	1

fig. A.2

Appendix B

Eulerian and Hamiltonian Circuits and Cycles

Eulerian Cycles and Circuits

Given a nondirected (directed) connected s, graph G an Eulerian cycle (circuit) is a simple cycle (circuit) which traverses every edge of G once and only once. An s-graph differs from a graph in that there may be as many as s edges connecting two vertices. We are interested then to know under what conditions Eulerian cycles exist.

Theorem 9: A connected, non directed s-graph G contains an Eulerian cycle if and only if the number of vertices of odd degree is zero.

Proof:

Necessity: Any Eulerian cycle must have one edge leaving a vertex and a different edge arriving at the same vertex since edges can be traversed only once. Hence, if G contains an Eulerian cycle, the degrees of all vertices must be even.

Sufficiency: Let G have vertices of even degrees only. Starting from a vertex v_i we build a path comprising of edges that have not been previously used, until we return to vertex v_i again. If all edges have been used, then we would have formed an Eulerian cycle. Otherwise let C_1 denote the cycle we have first formed. Since G is connected, there is at least one vertex $v_j \varepsilon C_1$ where v_j is ther terminal or initial vertex of some edge, the subgraph obtained by removing C_1 from G is

a connected graph where the vertices have even degrees, since C_1 uses only an even number of edges incident to its vertices. Another cycle can be generated using the same procedure as above starting from v_j on the subgraph. If all edges are used then let us denote the new cycle by C_2. A simple cycle $C_1 \cup C_2$ denotes an Eulerian cycle having a common vertex at v_j.

If some edges are left then we find $v_k \in C_1 \cup C_2$ such that v_k is the terminal or initial vertex of an edge and repeat the process. We do this until all edges are considered. The union of all the cycles formed in this way corresponds to our Eulerian cycle. ■

An important consideration is that G is connected. If not then we would incur components of the graph which are not connected and thus no Eulerian cycle can be formed that includes all the edges.

Collorary: A connected directed s graph G contains an Eulerian circuit if and only if the indegree of a vertex is equal to the outdegree of the same vertex for all vertices in the graph.

If we represent an undirected graph by an adjacency matrix $A = [(a_{ij})]$ where (a_{ij}) is the number of edges from i to j, if edges (i,j) exist and 0 otherwise, the sum of the rows and columns of this matrix would tell us the in degree and out degree of each vertex, respectively. The sum of the i^{th} row and i^{th} column would denote the degree of vertex i.

The method to obtain an Eulerian circuit is now explained - very simple:

Start from any vertex x_i and select an edge to traverse such that if deleted, the graph does not become two unconnected components.

Hamiltonian Cycles and Circuits

A Hamiltonian cycle (circuit) is an elementary cycle (circuit) that contains all the vertices in the graph. Although the problem resembles that of finding Eulerian cycles (circuits), it is very different and only partial results for special classes of graphs exist. Some special results are listed here for interest:

1. For a complete graph, where each vertex is connected to every other vertices, there are an enormous number of Hamiltonian cycles (circuits).
2. No bipartite graph having odd number of vertices possesses a Hamiltonian cycle (circuit). A bipartite graph G(N,A) is one where we could partition the set of N into N_1, N_2 such that $N_1 \cup N_2 = N$ and $N_1 \cap N_2 = \phi$ and no edges connect vertices belonging to the same set of vertices, i.e. for $(i,j) \in A$, $i \in N_i$, $j \in N_2$ or $i \in N_2$, $j = N_1$. From this definition, we know that every elementary cycle in a bipartite graph has an even number of edges and hence is incident to an even number of vertices.
3. By definition of strong connectivity, if a Hamiltonian cycle (circuit) exists, then the graph is strongly connected. The reverse is not true. If two strong circuits exist, then Hamiltonian circuit cannot exist.

Given an undirected graph, a Hamiltonian cycle might or might not exist. For the case of directed graphs we would expect the existence of a Hamiltonian circuit to be rarer since orientation of edges also needs to be taken into account. If we assign an edge cost of one unit to every edge in the graph, then the elementary cycle (circuit) whose cost is n would constitute a Hamiltonian cycle (circuit) i.e., if there are n vertices in the graph, then there must be n edges forming the Hamiltonian cycle (circuit). Using any of the methods discussed in Section 2, we would be able to enumerate the Hamiltonian cycles (circuits) if any exist. However, this would be rather tedious.

Thus far, two methods have been proposed that provide some improvements over the enumerative methods discussed in Section 2. These could be categorized under algebraic and enumerative procedures. These methods determine whether any Hamiltonian circuits exist and if so, to enumerate them. The algebraic method is based on the work by Yau [37], Danielson [8] and Dhawan [11] and involves the generation of all elementary paths by successive matrix manipulation. As such, it incurs much storage since it would have to store all paths that might conceivably form part of all Hamiltonian circuits.

The enumerative procedure considers one path at a time, which is continuously extended until such time as: either a Hamiltonian circuit is formed, or it becomes apparent that this path will not lead to a Hamiltonian circuit. The path is then modified to ensure that all possibilities will be included and the search continues. This procedure uses less storage and finds Hamiltonian circuits one at a

time (as opposed to the first method which attempts to find all Hamiltonian circuits at once). This method was suggested by Roberts and Flores [29].

Method A: Algebraic Method

Define $B = [b(i,j)]$ as an n x n matrix where $b(i,j) = j$ if arc (i,j) exists and zero otherwise. We define the internal vertex product of a path as the sequence of vertices of the path excluding the initial and terminal vertices. We next define $B_\ell = [b_\ell(i,j)]$ as an n x n matrix where $b_\ell(i,j)$ is the sum of the internal vertex products of all elementary paths of length ℓ from i to j where $i \neq j$. Assume $b_\ell(i,j) = 0$ for all i and A as the adjacency matrix. The matrix multiplication $B \cdot B_\ell = B'_{\ell+1} = [b_{\ell+1}(s,t)]$, where $B'_{\ell+1}(s,t) = \sum_k b(s,k) \cdot b_\ell(k,t)$. $B_{\ell+1}(s,t)$ then is the sum of the inner products of all paths from s to t of length $\ell + 1$. Since $b_\ell(k,t)$ represents all elementary paths of length ℓ from k to t, non elementary paths can appear only when the inner vertex product contains vertex s. Thus, if all terms containing s are eliminated from $B'_{\ell+1}(s,t)$, we would get a matrix $B_{\ell+1} = [b_{\ell+1}(s,t)]$ where if we set all diagonal elements to 0, then $B_{\ell+1}(s,t)$ gives us all elementary paths of cardinality $\ell + 1$ from s to t. Having $B_{\ell+1}$ we continue to take $B \cdot B_{\ell+1}$ to get $B_{\ell+2}$ after the proper reduction, etc., until a matrix B_{n-1} is generated. This matrix gives us all Hamiltonian paths between pairs of vertices. We could then identify Hamiltonian circuits from the paths of B_{n-1} since we need only to see if an arc

connects the terminal vertex to the initial vertex. Otherwise, we could take $B \cdot B_{n-1}$ to identify all Hamiltonian circuits from the non zero diagonal elements. An example is provided in Appendix B1.

A slight modification of the above method reduces both the storage and running time. Since we are interested only in Hamiltonian circuits, we could obtain this from the diagonal elements of the matrix $B \cdot B_{n-1}$. We note here that since only the element $B_n(1,1)$ is needed, it is necessary only to store the first column of B_ℓ, and eliminating non elementary elements we could obtain the first column of $B_{\ell+1}$. Continuing, we could obtain the first column of B_{n-1}. The matrix multiplication B and the first column of B_{n-1}, would give us the required Hamiltonian circuit if any exists.

Method B: Enumerative Procedure

We shall first present the algorithm of Roberts and Flores and then show how we could make improvements for graphs with more than 20 vertices.

Define a k x n matrix $M = [m_{ij}]$ where m_{ij} is the i^{th} vertex (x_q say) for which an arc (x_j, x_q) exists in a graph $G(V,\Gamma)$. The vertices $x_q \in \Gamma(x_j)$ are compacted to form entries in the j^{th} column of the M matrix. The number of rows k corresponds to the maximum outdegree of a vertex. See Appendix B2.

Choose an initial vertex x_1. x_1 is the first entry in set S which represents a vertex sequence of elementary path. Build the path by adding the first element (a say) in the x_1 column to S. Proceed by

extending the first feasible vertex in the a^{th} column to S. A vertex is feasible if it has not already been in S. Continue to extend the path until no further extension is possible. Let $S = \{x_1,a,b,c,\ldots,x_{r-1},x_r\}$. Then either no vertex in column x_r is feasible or the path has cardinality n - 1. In the latter case, if there is an arc (x_r,x_1) then a Hamiltonian circuit is identified. Otherwise, no Hamiltonian circuit can be formed using this path.

In any case, we backtrack. Backtracking involves removing x_r from S and adding the first feasible vertex below x_r to S in column x_{r-1}. If no feasible vertex is formed, we backtrack further. The algorithm terminates when x_1 is the only element in S and backtracking leaves S empty. All Hamiltonian circuits in the graph have been identified. Consider the simple example in Appendix B2.

In our path extension two conditions would enable us to build a path more effectively. Let us assume that at some point during the search the path is given by $S = \{x_1,x_2,\ldots,x_r\}$, and the next vertex to be included is $x^* \notin S$. Let the original directed graph be $G(V,\Gamma)$, and x be a vertex of the subgraph obtained by deleting all the vertices in S from V.

1. If there exists a vertex $x \in V - S$ such that $x \in \Gamma(x_r)$, i.e. (x_r,x) exists, and $\Gamma^{-1}(x) \subseteq S$, i.e. the set of all vertices with arcs directed towards x are contained in S, then the extension vertex is x, i.e. next arc is (x_r,x) where $x=x^*$. If a different vertex is chosen, then any path formed subsequently would not include x and therefore cannot be a Hamiltonian

path. See fig. B.1.

2. If there exists a vertex $x \in V - S$ such that, $x \notin \Gamma^{-1}(x_1)$ i.e. arc (x_1,x) does not exist, and $\Gamma(x) \subset S \cup \{x^*\}$, for some other vertex x^*, i.e. set of vertices with arcs directed from x must be contained in S or in $\{x^*\}$, then x^* cannot be included as the next vertex in the path since no paths between x and x_1 would be possible. Another arc other than (x_r,x^*) must be considered. (See fig. B.2.)

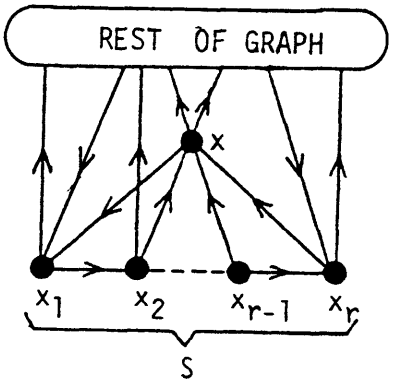


fig. B.1

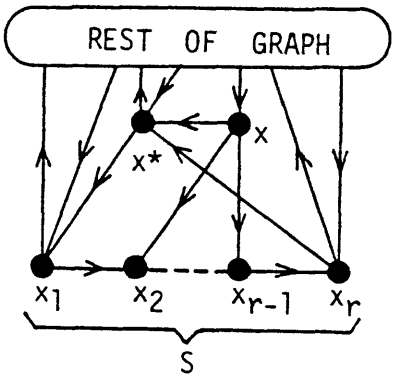


fig. B.2

In the example of Appendix B2, condition 1 arises at step 2 when $S = \{a,b\}$. For vertex e, $\Gamma^{-1}(e) = \{b\} \subset S$ so that e must be the next vertex. Steps 3 to 8 could be skipped.

Given a weighted graph with cost associated with the arcs, the general traveling salesman problem attempts to find the minimum cost Hamiltonian cycle (circuit). For a discussion of the Traveling Salesman problem the reader is referred to Christofides [7].

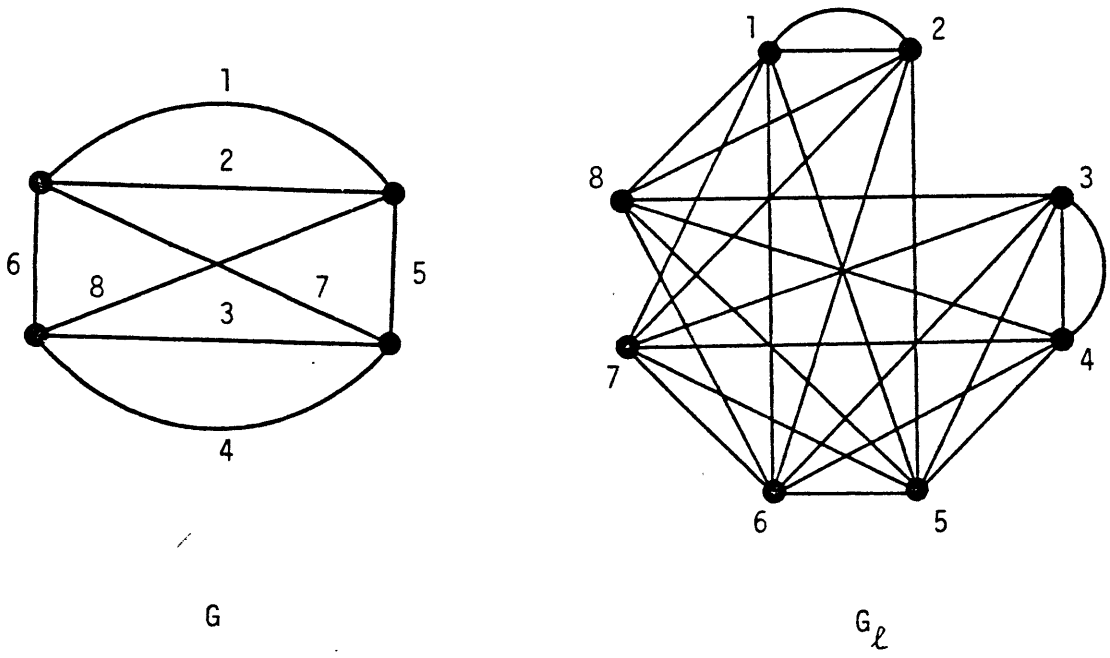
Although the improved version of Robert and Flores [29] procedure would reduce running time by half for a randomly generated graph of more than 20 vertices, another method is available for a large scale graph which is superior. It reduces the graph by searching for paths that cannot possibly belong to a Hamiltonian cycle or circuit. The set of paths S considered in Roberts and Flores [29] does not adequately consider the effects of path extension S on the remaining portions of the graph. A detailed account of this reduction on the graph so that Hamiltonian cycles (circuits) would be more effectively searched is found in Christofides [7] and is called the Multi Path method. Although the methods discussed thus far apply to directed graphs, extension to the undirected case is also possible.

Relationship between Eulerian and Hamiltonian Cycles (Harary [B5])

For an undirected graph, the definitions of Eulerian and Hamiltonian cycles permit us to establish a relationship between them. We define a line graph of G_ℓ of G as a graph having as many vertices as there are edges in the graph G. An edge between two vertices of G_ℓ exists if and only if the edges of G corresponding to these two vertices are adjacent (i.e. incident to the same vertex in G).

Consider the graph G and its corresponding line graph G_ℓ in fig. B.3.

fig. B.3



G G_ℓ

Note: Vertices in G and G_ℓ have even degree. (Eulerian cycle exists)
(1,2,5,4,8,3,6,7,1) is a Hamiltonian cycle in G_ℓ.

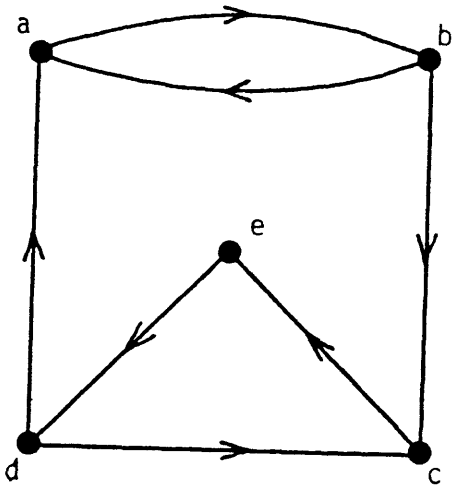
It can be easily shown that: -

1. If G has a Eulerian cycle, the G_ℓ has both Eulerian and Hamiltonian cycles.
2. If G has a Hamiltonian cycle, then G_ℓ has a Hamiltonian cycle. The converse of both statements are not true.

Appendix B1

Enumeration of Hamiltonian Circuits Using the Algebraic Approach: An Example (Christofides [7])

Given



$A =$

	a	b	c	d	e
a	0	1	0	0	0
b	1	0	1	0	0
c	0	0	0	0	1
d	1	0	1	0	0
e	0	0	0	1	0

Let $B_1 = A$

$B =$

	a	b	c	d	e
a	0	b	0	0	0
b	a	0	c	0	0
c	0	0	0	0	e
d	a	0	c	0	0
e	0	0	0	d	0

$B_2' = B \cdot B_1$

	a	b	c	d	e
a	<u>b</u>	0	b	0	0
b	0	<u>a</u>	0	0	c
c	0	0	0	e	0
d	0	a	0	0	c
e	d	0	d	0	0

2-paths from i to j via vertex listed in (i,j). B_2 is similar to B_2' with elements underlined set to zero.

$B_3' = B \cdot B_2 =$

	a	b	c	d	e
a	0	0	0	0	bc
b	0	0	<u>ab</u>	ce	0
c	ed	0	<u>ed</u>	0	0
d	0	0	ab	<u>ce</u>	0
e	0	da	0	0	<u>dc</u>

3-paths from i to j via vertex listed in (i,j). P_3 is similar to B_3' with elements underlined set to zero. Note the entry for paths from b to c contains b once again and is eliminated as non-elementary.

$B_4' = B \cdot B_3 =$

	a	b	c	d	e
a	0	0	0	bce	0
b	ced	0	0	0	<u>abc</u>
c	0	eda	0	0	0
d	<u>ced</u>	0	0	0	abc
e	0	0	dab	0	0

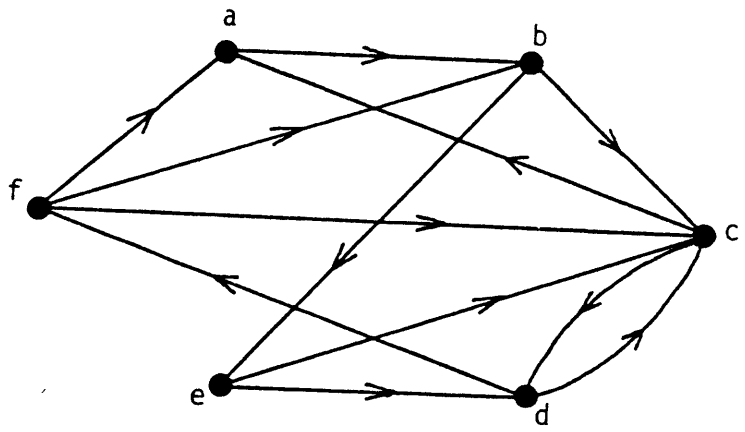
4-paths from i to j via vertex listed in (i,j). B_4 is similar to B_4' with elements underlined set to zero.

B_4 contains all the elementary 4-paths from i to j. A Hamiltonian path is given by $b_4(1,4)$, which is abced. Since from B, b(4,1) = a, i.e., there is an arc from d to a, abceda is the required Hamiltonian circuit.

Appendix B2

Enumeration of Hamiltonian Circuit Using Enumerative Procedure: An Example (Christofides [7])

Given:



$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{bmatrix} b & c & a & c & c & a \\ & e & d & f & d & b \\ & & & & & c \end{bmatrix} \end{array}$$

k is equal to 3 corresponding to the maximum outdegree of vertex f.

The search to find all Hamiltonian circuits now proceeds as follows: (Vertex a is taken as the starting vertex.)

	Set S		Notes
1.	a	:	Add first feasible vertex in column a (i.e. vertex b)
2.	a,b	:	Add first feasible vertex in column b (i.e. vertex c)
3.	a,b,c	:	First vertex (a) in column c is infeasible (a ε S), add next vertex in column c (i.e. vertex d)
4.	a,b,c,d	:	Add vertex f
5.	a,b,c,d,f	:	No feasible vertex in column f exists. Backtrack. Remove f from S. Close f in column d.
6.	a,b,c,d	:	No feasible vertex in column d exists. Backtrack. Remove d from S. Close d in column c.
7.	a,b,c	:	Similar to case above. Backtrack. Remove c from S. Close c in column b.
8.	a,b	:	Add vertex e. Open all vertices.
9.	a,b,e	:	Add vertex c
10.	a,b,e,c	:	Add vertex d
11.	a,b,e,c,d	:	Add vertex f
12.	a,b,e,c,d,f	:	Hamiltonian path. Hamiltonian circuit can be closed by arc(f,a). Backtrack
13.	a,b,e,c,d	:	Backtrack
14.	a,b,e,c	:	Backtrack
15.	a,b,e	:	Add vertex d
16.	a,b,e,d	:	Add vertex f
17.	a,b,e,d,f	:	Add vertex c
18.	a,b,e,d,f,c	:	Hamiltonian path. Circuit closed by arc(c,a). Backtrack c is closed.

	Set S		Notes
19.	a,b,e,d,f	:	Backtrack f is closed.
20.	a,b,e,d	:	Backtrack d is closed.
21.	a,b,e	:	Backtrack e is closed.
22.	a,b	:	Backtrack b is closed.
23.	a	:	Backtrack
24.	Ø	:	End of search.

References

[1] M.T. Ardon and N.R. Malik, A recursive algorithm for generating circuits and related subgraphs, 5th Asilomar Conf. on circuits and systems 5, Pacific Grove, California, Nov. 1971, pp. 279-284.

[2] A.T. Berztiss, Data Structures: Theory and Practice, Academic Press, NY, 1971.

[3] D. Cartwright and T.C. Gleason, The number of paths and cycles in a digraph, Psychometrika, 31 (1966), pp. 179-199.

[4] S.G. Chan and W.T. Chang, On the determination of dicircuits and dipaths. Proc. 2nd Hawaii Internat. Conf. System Sci., Honolulu, Hawaii 1969, pp. 155-158.

[5] J.P. Char, Master circuit matrix, Proc. IEE (London) 115 (1968), pp. 762-770.

[6] N. Christofides, Graph Theory: An Algorithmic Approach, Academic Press, 1975.

[7] S.M. Chase, Analysis of algorithms for finding all spanning trees of a graph, Ph.D. thesis, Rep. 401, Dept. of Computer Sci., U. of Illinois, Urbana, Illinois, 1970.

[8] G.H. Danielson, On finding the simple paths and circuits in a graph, IEEE Trans. Circuit Theory, CT-15 (1968), pp. 294.

[9] G.B. Dantzig, W.O. Blattner and M.R. Rao, Finding a cycle in a graph with minimum cost to time ratio with applications to ship routing problem, in Int. Symp. on Theory of Graphs, Dunod, Paris, (1966), pp. 77.

[10] Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science, Prentice Hall, 1974.

[11] V. Dhawan, Hamiltonian circuit and related problems in graph theory, M.Sc. Report, Imperial College, London (1969).

[12] A. Ehrenfeucht, L. Fosdick and L. Osterweil, An algorithm for finding the elementary circuits of a directed graph, Tech. Rep. CU-CS-024-73, Dept. of Computer Sci., U. of Colorado, Boulder, 1973.

[13] R.W. Floyd, Nondeterminstic algorithms, JACM, 14 (1967), pp. 636-644.

[14] N.E. Gibbs, A cycle generation algorithm for finite undirected linear graphs, JACM 16 (1969), pp. 564-568.

[15] C.C. Gotlieb and Corniel, Algorithms for finding a fundamental set of cycles for an undirected linear graph, Comm. ACM 10 (1967), pp. 780-783.

[16] J.E. Hoftcroft and R.E. Tarjan, Efficient planarity testing, J ACM 21 (1974), pp. 549-558.

[17] H.T. Hsu and P.A. Honkanen, A fast minimal storage algorithm for determining all the elementary cycles of a graph, Computer Sci. Dept., Penn. State U., University Park, (1972).

[18] D.B. Johnson, Finding all the elementary circuits of a directed graph, SIAM J. Comp. 4 (1975), pp. 77-84.

[19] T. Kamae, A systematic method of finding all directed circuits and enumerating all directed paths, IEEE Trans. Circuit Theory, CT-14 2 (1967), pp. 166-171.

[20] J.W. LaPatra and B.R. Meyers, Algorithms for circuit enumeration, IEEE International Conv. Record, Part 1, 12 (1964), pp. 368-373.

[21] E.L. Lawler, Optimal cycle in doubly weighted graphs, in Int. Symp. on Theory of Graphs, Dunod, Paris (1966), p. 209.

[22] C.L. Liu, Introduction to Combinatorial Mathematics, McGraw-Hill, 1968.

[23] P. Mateti and N. Deo, On algorithm for enumerating all circuits of a graph, SIAM J. Comput., Vol. 5, No. 1, March 1976.

[24] L.M. Maxwell and G.B. Reed, Subgraph identification - Segs, circuits and paths, 8th Midwest Symp. on Circuit Theory, Colorado State U., Fort Collins, Colorado, June 1965, pp. 13-0-13-10.

[25] K. Paton, An algorithm for finding a fundamental set of cycles for an undirected linear graph, Comm. ACM 12 (1969), pp. 514-518.

[26] J. Ponstein, Self-avoiding paths and adjacency matrix of a graph, SIAM J. Appl. Math. 14 (1966), pp. 600-609.

[27] V.V.B. Rao and V.G.K. Murti, Enumeration of all circuits of a graph, Proc. IEEE 57 (1969), pp. 700-701.

[28] R.C. Read and R.E. Tarjan, Bounds on backtrack algorithms for listing cycles, paths and spanning trees, Networks: ERL Memo M433, Electronics Research Lab., U. of California Berkeley, 1973.

[29] S.M. Roberts and B. Flores, Systematic generation of Hamiltonian circuits, Comm. ACM 9 (1966), pp. 690-694.

[30] R.W. Simpson, Scheduling and routing models for airline systems. Report FTL-R68-3, Dept. Aeronautics and Astronautics, MIT, (1968).

[31] J.L. Szwarcfiter and P.E. Lauer, Finding the elementary cycles of a directed graph in O(n + m) per cycle, No. 60, U. of Newcastle upon Tyne, England, May 1974.

[32] R. Tarjan, Enumeration of the elementary circuits of a directed graph, SIAM J Comp. 2 (1973), pp. 211-216.

[33] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Computing 1 (1972), pp. 146-160.

[34] J.C. Tiernan, An efficient search algorithm to find the elementary circuits of a graph, Comm. ACM 13 (1970), pp. 722-726.

[35] H. Weinblatt, A new search algorithm for finding the simple cycles of a finite directed graph, JACM 19 (1972), pp. 43-56.

[36] J.T. Welch, A mechanical analysis of the cyclic structure of undirected linear graphs, JACM 13 (1966), pp. 205-210.

[37] S.S. Yau, Generation of all Hamiltonian circuits, paths, and centers of a graph, and related problems, IEEE Trans. Circuit Theory, CT-14 (1967), pp. 79-81.

Bibliography

[B1]. C. Berge, The Theory of Graphs and Its Applications, Methuen, Translated by A. Doig from original French edition, Dunod, 1962.

[B2] R.G. Busacker and T.L. Saaty, Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill, 1965.

[B3] N. Christofides, Graph Theory: An Algorithmic Approach, Academic Press, London 1975.

[B4] N. Deo, Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall, N.J. 1974.

[B5] F. Harary, Graph Theory, Addison-Wesley, Reading, MA, 1972.

[B6] C.L. Liu, Introduction to Combinatorial Mathematics, McGraw-Hill, 1968.

[B7] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness. San Francisco: W.H. Freeman & Co., Publishers 1979.