An Efficient Search Algorithm to Find the Elementary Circuits of a Graph

JAMES C. TIERNAN
*University of California, San Diego**
La Jolla, California

A theoretically most efficient search algorithm is presented which uses an exhaustive search to find all of the elementary circuits of a graph. The algorithm can be easily modified to find all of the elementary circuits with a particular attribute such as length. A rigorous proof of the algorithm is given as well as an example of its application. Empirical bounds are presented relating the speed of the algorithm to the number of vertices and the number of arcs. The speed is also related to the number of circuits in the graph to give a relation between speed and complexity. Extensions to undirected and s-graphs are discussed.

KEY WORDS AND PHRASES: algorithm, graph theory, circuit search algorithm, path search algorithm, searching
CR CATEGORIES: 3.74, 5.32

Introduction

Several articles have recently been published dealing with finding circuits or cycles in a graph [2, 4, 5, 6, 7]. The approach, in general, has been to consider undirected graphs and find a fast algorithm for generating the fundamental set of cycles of the graph. All linear combinations of the fundamental set then give the total set of cycles. In this paper attention is focussed on directed graphs where one can no longer simply use the linear space properties of undirected graphs. With directed graphs, search methods prove appealing. The search algorithm presented here considers each circuit only once during the search. Since in any search algorithm each circuit must be considered at least once, this property makes the present algorithm the theoretically most efficient search algorithm in basic structure. It is immediately much better than any algorithm which finds circuits of a fixed length and is iteratively used to find all circuits. This approach causes small circuits to be considered many times.

The form of presentation of the algorithm is neither a flowchart nor an Algol program. The author believes that the method of algorithm presentation advanced by Knuth is superior to these and is worth giving a trial in publication [3].

Background

A graph is defined as a set of vertices, $V = \{v_1, v_2, \cdots, v_N\}$, and an *arc operator* $\Gamma(\cdot)$ which operates on all the individual elements of V to give subsets of V [1]. In the pictorial form of a graph each vertex is represented by a point and the relation $v_j \in \Gamma(v_i)$ is represented by an *arc* (a directed line) from the point v_i to the point v_j. In this context v_i is the initial vertex of the arc and v_j is the terminal vertex of the arc. For example, consider Figure 1, where

$$\begin{aligned} V &= \{v_1, v_2, \cdots, v_5\} \\ \Gamma(v_1) &= \{v_2\} \\ \Gamma(v_2) &= \{v_2, v_3, v_4\} \\ &\vdots \\ \Gamma(v_5) &= \{v_1\} \end{aligned}$$

A *path* is a sequence of arcs with the terminal vertex of an arc in the sequence as the initial vertex of the next arc in the sequence. A path is specified by the sequence of vertices one would encounter while moving along the path. In Figure 1 an example path is $[v_1, v_2, v_2, v_3, v_5]$.

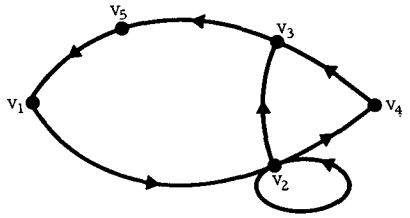

FIG. 1. Pictorial form of a graph

An *elementary path* contains each vertex at most once in its specification.

An *elementary circuit* is an elementary path with the exception that the first and last vertices of the path are the same.

The Algorithm EC

The algorithm is named EC for "elementary circuit." The function blocks of EC are denoted EC1, EC2, etc.

EC requires that there be assigned to the vertices, in any order, the integer designators 1, 2, $\cdots$, N. The algorithm utilizes two principal arrays in addition to that describing the graph. The first is a one-dimensional array, P, containing the vertices of an elementary path. The second is a two-dimensional array, H, and is initially zeroed. Elementary path building in P is the basic process of EC. The first path is started as vertex 1. A path is extended from its end, one arc at a time, with three conditions checked before a tentative extension is performed:

1. The extension vertex cannot be in P.
2. The extension vertex value must be larger than that of the first vertex of P.
3. The extension vertex cannot be *closed* to the last vertex in P. H contains the list of vertices closed to each vertex.

Condition 1 assures that an elementary path is being formed. Condition 2 assures that each circuit will only be considered once. The search for a particular circuit will

* Department of Applied Physics and Information Science

only occur when its lowest valued integer is at the path beginning. Condition 3 assures that no elementary path is considered more than once.

Consider the extension process as the path snakes out from vertex 1. Condition 1 and, of course, the constraint of remaining in the graph are the only limits to initial path building. However, at some point no vertices will be *available for extension*. A *circuit confirmation* is then performed; i.e. is there an arc connecting the last vertex of P to the first vertex? If there is, a circuit is reported. In any case, *vertex closure* occurs unless there is only one vertex in the path. The vertex closure process 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 eliminating the last vertex.

Step 1 assures that the path extension just performed will not be repeated. Step 2 allows correct forward continuation from the last vertex if it is reached by a different path in the future. The above extension process continues until eventually the path has been backed down to vertex 1. At this point an *initial vertex advancement* occurs. In this step, the first vertex is incremented by one, H is cleared,

P	*Action on attained path*
1 0 0 0 0	
1 2 0 0 0	
1 2 3 0 0	
1 2 3 5 0	report circuit, H[3, 1] ← 5.
1 2 3 0 0	no circuit, H[3, 1] ← 0, H[2, 1] ← 3.
1 2 0 0 0	
1 2 4 0 0	
1 2 4 3 0	
1 2 4 3 5	report circuit, H[3, 1] ← 5.
1 2 4 3 0	no circuit, H[3, 1] ← 0, H[4, 1] ← 3.
1 2 4 0 0	no circuit, H[4, 1] ← 0, H[2, 2] ← 4.
1 2 0 0 0	no circuit, H[2, 1] ← 0, H[2, 2] ← 0, H[1, 1] ← 2
1 0 0 0 0	no circuit, advance P[1], clear H.
2 0 0 0 0	
2 3 0 0 0	
2 3 5 0 0	no circuit, H[3, 1] ← 5.
2 3 0 0 0	no circuit, H[3, 1] ← 0, H[2, 1] ← 3.
2 0 0 0 0	
2 4 0 0 0	
2 4 3 0 0	
2 4 3 5 0	no circuit, H[3, 1] ← 5.
2 4 3 0 0	no circuit, H[3, 1] ← 0, H[4, 1] ← 3.
2 4 0 0 0	no circuit, H[4, 1] ← 0, H[2, 1] ← 4.
2 0 0 0 0	report circuit, advance P[1], clear H.
3 0 0 0 0	
3 5 0 0 0	no circuit, H]3, 1] ← 5.
3 0 0 0 0	no circuit, advance P[1], clear H.
4 0 0 0 0	no circuit, advance P[1], clear H.
5 0 0 0 0	no circuit, terminate.

Circuits Reported
1 2 3 5
1 2 4 3 5
2

FIG. 2. Evaluation of elementary circuits of graph in Figure 1

and the extension process starts anew. Now, however, no paths, and thus circuits, containing vertex 1 will be considered. EC continues to extend paths and, on occasion, advance the initial vertex until P contains a path of one vertex, vertex N. Then termination takes place.

EC is started by specifying the graph (V, Γ). Each vertex is given a number, $V = \{1, 2, \cdots, N\}$.

The Γ-relations are presented in an $N \times N$ array, $G[i, j]$. The entries in row i are the elements in $\Gamma(i)$. These are entered sequentially from left to right in the row. Thus, the first zero indicates the end of the list for $\Gamma(i)$.

The output of EC is a listing of circuits. Each circuit is given in vertex sequence form, $[v_1, v_2, \cdots, v_K]$, $K \leq N$. The initial vertex is not repeated as the terminal vertex since this is redundant. The circuits are outputted as they are found.

The variables of EC are as follows:

$P[1]$ is an $N \times 1$ array. P contains the vertex sequence of the present path under consideration. $P[1]$ is the first vertex of the sequence and $P[k]$ is the last.

$H[n, m]$ is an $N \times N$ array. Entries in row n are closed to vertex n. The entries are made sequentially from left to right in the row. Thus, the first zero in a row indicates the end of the closure list.

In the presentation of EC below each significant operation is given a numeric label for ease of reference. The action flows from one function block to the next unless a "go to" statement is encountered. The example of Figure 2 can be used to help follow the algorithm.

Algorithm EC

EC1: [Initialize]
1: Read N and G.
$P \leftarrow 0$
$H \leftarrow 0$
$k \leftarrow 1$
$P[1] \leftarrow 1$
EC2: [Path Extension]
2: Search $G[P[k], j]$ for $j = 1, 2, \cdots, N$ such that the following three conditions are satisfied,
(1) $G[P[k], j] > P[1]$
(2) $G[P[k], j] \notin P$
(3) $G[P[k], j] \notin H[P[k], m]$, $m = 1, 2, \cdots, N$.
3: If this j is found, extend the path,
$k \leftarrow k + 1$
$P[k] \leftarrow G[P[k - 1], j]$
go to EC2.
4: If no j meets the above conditions, the path cannot be extended.
EC3: [Circuit Confirmation]
5: If $P[1] \notin G[P[k], j]$, $j = 1, 2, \cdots, N$ then no circuit has been formed, go to EC4.
6: Otherwise a circuit is reported,
Print P.
EC4: [Vertex Closure]
7: If $k = 1$, then all of the circuits containing vertex $P[1]$ have been considered.
go to EC5.
8: Otherwise,
$H[P[k], m] \leftarrow 0$, $m = 1, 2, \cdots, N$

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 EC2.

EC5: [Advance Initial Vertex]

9: If $P[1] = N$ then,
go to EC6.

10: Otherwise,
$P[1] \leftarrow P[1] + 1$
$k \leftarrow 1$
$H \leftarrow 0$
go to EC2.

EC6: [Terminate]

Proof of Algorithm

An algorithm has five features: (1) definiteness, (2) input, (3) output, (4) finiteness, and (5) effectiveness [3]. To this point EC has been defined and the input/output has been specified. Thus it remains to be proved that EC is finite and effective. Finiteness is simply the property that the functional block EC6 will be attained by the algorithm in a finite number of steps. It is claimed that EC finds and prints every elementary circuit of the inputted graph and does this only once for each circuit. If this is indeed the case, then EC is effective.

THEOREM 1. *Algorithm EC is finite.*

PROOF. Assume EC is infinite. This implies that there is at least one vertex number a such that for $P[1] = a$ an infinite number of steps are performed; i.e. the condition of EC4 line 7 is never met and EC5 is not reached for $P[1] = a$. Due to conditions (1) and (2) of EC2 line 2, only $N - a$ vertices can be open to a for extension. After extension to at least one of the vertices open to a, call it b, the algorithm must perform an infinite number of steps. For if only a finite number of steps are performed in each vertex open to a, a itself must be finite. Due to conditions (1) and (2) of EC2 line 2, only $N - a - 1$ vertices can be open to b for extension. Let the extensions process of EC2 continue for m steps. At each extension, that vertex is chosen which requires EC to perform an infinite number of steps. After m steps there are at most $N - a - m$ vertices possibly open to the last vertex, one at least which must require an infinite number of steps for exhaustion. But, for $m = N - a$ there can be no vertices open to the last vertex by the conditions of EC2 line 2. This is a contradiction. Thus, EC is finite. |

A path will be defined as *attained* if the algorithm enters EC3 from EC2 with P set to this path vertex sequence. It can be seen that for all paths attained, $[v_1, v_2, \cdots, v_K]$, it is true that $v_1 < v_2, v_3, \cdots, v_K$ because of the conditions of EC2 line 2, applied to forming paths.

LEMMA 1. *If path* $[v_1, v_2, \cdots, v_K]$ *has been attained, then path* $[v_1, v_2, \cdots, v_L]$ *which is properly contained in* $[v_1, v_2, \cdots, v_N]$ *will be attained.*[1]

[1] Path A properly contains path B if and only if B can be extended to A but A cannot be extended to B in EC2.

PROOF. If $[v_1, v_2, \cdots, v_K]$ has been attained, then in EC4 the path $[v_1, v_2, \cdots, v_{K-1}]$ is set to be extended in EC2. In EC2 a path will be attained which contains $[v_1, v_2, \cdots, v_{K-1}]$. If the containment is proper, then in the next extension cycle a path will be extended which contains $[v_1, v_2, \cdots, v_{K-1}]$ because only one vertex is eliminated in EC4. Thus one must come to an extension which attains $[v_1, v_2, \cdots, v_{K-1}]$ since EC is finite, i.e. a path which contains $[v_1, v_2, \cdots, v_{K-1}]$ but does not properly contain it must be attained.

Now assume that $[v_1, v_2, \cdots, v_{L+1}]$ is attained. By the same argument as for $[v_1, v_2, \cdots, v_{K-1}]$ the path $[v_1, v_2, \cdots, v_L]$ is attained and the Lemma 1 is proved by induction. |

THEOREM 2. *Every elementary path of the following form will be attained by EC:* $[v_1, v_2, \cdots, v_K]$, $v_1 < v_2, v_3, \cdots, v_K$, $K \leqq N$.

PROOF. The theorem will be proved by induction. Consider an arbitrary path, $[v_1, v_2, \cdots, v_K]$, $v_1 < v_2, v_3, \cdots, v_K$. Let $K = 1$. In EC1, P is initialized to $P = [1]$ and extended in EC2. Thus a path containing $P = [1]$ is attained and by Lemma 1, $P = [1]$ is attained. By Theorem 1, EC is finite. Thus EC5 will be reached and $P = [2]$, $P = [3]$, $\cdots$, $P = [N]$ will be extended in EC2 in turn. Thus each of these will be attained by Lemma 1. Since $P = [v_1]$ is among them, it will be attained.

Assume the theorem is true for all $K \leqq M - 1$ and let us show that it is true for $K = M$. Thus we wish to show that the arbitrary path $[v_1, v_2, \cdots, v_M]$, $v_1 < v_2, v_3, \cdots, v_M$, is attained when $[v_1, v_2, \cdots, v_{M-1}]$ is attained. For $[v_1, v_2, \cdots, v_{M-1}]$ to be attained, the condition of EC2 line 4 must be satisfied with $P = [v_1, v_2, \cdots, v_{M-1}]$. But v_M satisfies conditions 1 and 2 of EC2 line 2 by assumption. This implies that when $[v_1, v_2, \cdots, v_{M-1}]$ is attained, $v_M \in H[v_{M-1}, m]$, $m = 1, 2, \cdots, N$. This can only occur in EC4 after $[v_1, v_2, \cdots, v_M]$ has been attained. |

COROLLARY. *Every elementary circuit will be printed by EC.*

PROOF. Consider an arbitrary circuit of the inputted graph, $[a_1, a_2, \cdots, a_K, a_1]$. This circuit can be expressed in a permuted form with the lowest integer specified vertex appearing first and also not repeated as the last vertex,

$$[v_1, v_2, \cdots, v_K],\ v_1 < v_2, \cdots, v_K.$$

By Theorem 2 the elementary path $[v_1, v_2, \cdots, v_K]$ is attained by EC. By Theorem 1 a finite number of steps proceed before all vertices in $\Gamma(v_K)$ are unavailable for extension. At this point EC2 line 4 occurs and since $v_1 \in \Gamma(v_K)$ the condition for EC3 line 6 is met and the circuit is printed. Since $[a_1, a_2, \cdots, a_K, a_1]$ is an arbitrary choice, all circuits are reported. |

THEOREM 3. *No elementary circuit is printed twice by EC.*

PROOF. Again, consider arbitrary circuit $[a_1, a_2, \cdots, a_K, a_1]$. The algorithm can only report the circuit in the permuted form,

$$[v_1, v_2, \cdots, v_K],\ v_1 < v_2, v_3, \cdots, v_K.$$

This is so because all elementary paths of P are of this form and the circuits are printed as P in EC3 line 6. Thus, it must be shown that this form of the circuit is not reported twice. More generally, it will be proved by induction that function block EC cannot be reached twice with the same P. Again, the generalized path $[v_1, v_2, \cdots, v_K]$, $v_1 < v_2, v_3, \cdots, v_K$ will be considered.

Let $K = 1$. When EC3 is reached with $P = [v_1]$, then the condition of EC4 line 7 is met and $P[1]$ is increased or the algorithm is terminated in EC5. Thus $P = [v_1]$ cannot be reconsidered in EC3. Now, assume that EC3 cannot be reached twice for paths of length less than or equal to $M - 1$. Let $K = M$ in the specified path. This path is attained once by Theorem 2. In a finite number of steps all extensions from v_M will have been considered and $[v_1, v_2, \cdots, v_M]$ is considered in EC3. Then, in EC4, v_M is closed to v_{M-1}. For the elementary path $[v_1, v_2, \cdots, v_M]$ to be considered in EC3, v_M would have to be reopened to v_{M-1} which occurs when v_{M-1} is closed to v_{M-2} in EC4. But, if $[v_1, v_2, \cdots, v_{M-1}]$ is reformed to be extended to v_M, then it must be reconsidered in EC3 itself. But this is not possible by the inductive assumption. Thus the elementary path $[v_1, v_2, \cdots, v_M]$ cannot be considered more than once in EC3. This implies that no circuit can be printed twice.

An Example

Let us evaluate the circuits of the graph in Figure 1 as an example of the application of EC. The inputs to the algorithm are:

$$N = 5$$

$$G = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 2 & 3 & 4 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The search for vertex j in EC2 line 2 will occur from left to right in G. Figure 2 presents the values of P and the actions taken as the algorithm progresses.

Execution Speed

The *complete graph* is used as the basic tool in evaluating the speed of EC. A complete graph is defined[2] as a graph such that for all $x, y \in V$, $x \in \Gamma(y)$. Figure 3 is an example of a complete graph. Every graph of N vertices is a subgraph of a complete graph with N vertices.

Given a graph with N vertices, what is the upper bound on execution time? Given a graph with M arcs, what is the bound? Both questions are answered by looking at the complete graph. For a given N, the complete graph with N vertices requires the most searching to find the elementary circuits. For a given M, the complete graph with $\sqrt{M}$ vertices requires more computation than any other graph with M arcs. (If M is not a square we look at the next largest integer which is a square for an upper bound on computation.) Figures 4 and 5 relate the speed (CPU execution time) of EC, denoted T_p, to the number of vertices and arcs of complete graphs for a given computer.[3] These values will be linear with respect to the computational speed of the computer used.

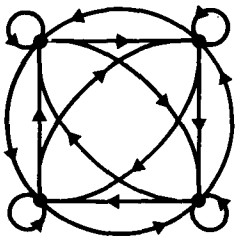

FIG. 3. A complete graph

The data of Figures 4 and 5 is all that is needed for practical evaluation of EC. However, one would like to have some measure of the speed of EC relative to the complexity of the graph being considered. In a sense, this gives absolute measure of the efficiency of EC. A good measure of the complexity of the graph is the number of circuits in the graph, T_N, since this is precisely the complexity that challenges us.

T_N is found by noting that each permutation of every arbitrary subset of the set of vertices $\{1, 2, \cdots, N\}$ represents a distinct circuit. These permutations can be partitioned according to their smallest numbered vertices—e.g. all permutations which have vertex 3 as their lowest numbered vertex form one class—Class 3. Consider the number of permutations in Class k. There are $(N - k)!/0!$ with $N - k + 1$ elements, $(N - k)!/1!$ with $N - k$ elements, and so forth to $(N - k)!/(N - k)!$ with one element (the loop). Thus, summing over all permutation classes,

$$T_N = \sum_{k=1}^{N} \sum_{i=0}^{N-k} \frac{(N-k)!}{i!}.$$

Letting $j = N - k$,

$$T_N = \sum_{j=0}^{N-1} j! \sum_{i=0}^{j} \frac{1}{i!}.$$

This can be bounded by using the Γ-function,

$$T_N < e \sum_{j=0}^{N-1} j! < e\left[1 + \int_1^N \Gamma(x+1)\, dx\right].$$

[2] This definition is not standard [1].

[3] The program was in Algol for the Burroughs B-5500 computer.

N	T_N	T_p (sec.)
1	1	0.12
2	3	0.13
3	8	0.15
4	24	0.30
5	89	0.80
6	415	4.20
7	2372	27.60
8	16072	219.00

FIG. 4. Comparison of execution speed bound T_p with number of circuits T_N and number of vertices N of a complete graph

T_N is included in Figure 4. It is seen that the algorithm behaves almost linearly in speed compared with T_N. This is an indication of the high efficiency of the algorithm. The relation between speed and complexity can be expressed an analytic bound on execution time T,

$$T < \eta\left[1 + \int_1^{\sqrt{M}} \Gamma(x+1)\,dx\right],$$

where η is a factor relating to computer computation speed. For the B-5500 computer, $\eta = 90$ is representative.

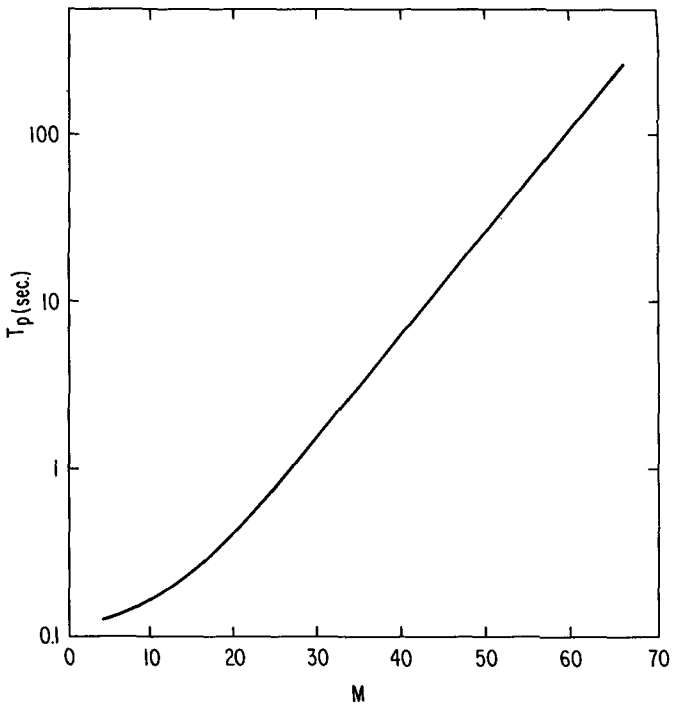

FIG. 5. Comparison of execution speed bound (T_p) with number of arcs (M).

Conclusion

The theoretically most efficient search algorithm for finding the elementary circuits of a graph has been presented. EC requires no special structure of the directed graph—for example, that it be planar. The algorithm can be used as a search procedure in any arbitrarily interconnected maze to find elementary paths as well as circuits with any desired attribute. As a matter of interest, the development of this algorithm was stimulated by an article on automata theory written by Traiger and Gill, which required a search algorithm to find a circuit with a particular attribute [6].

Several extensions of EC are immediate. The extension to undirected graphs requires counting each edge as an arc pair connecting the same two vertices, but with opposite orientation. This is accomplished in specifying the G-matrix. Additionally one must add a check for circuits of length 2 which, in actuality, are not cycles. EC finds each cycle twice when used for undirected graphs—once for each orientation of the circuit. The author has extended EC to undirected graphs. However, algorithms such as that by Paton do this job much better [4]. It is also of interest to extend EC to directed or undirected graphs with more than one arc or edge allowed to connect vertex pairs, i.e. s-graph [1]. This is accomplished by reducing every multiple arc set connecting a vertex pair to a single arc and attaching a number to this arc. For example, if three arcs go from i to j, we form one arc and let k_i^j be the number attached to this arc $k_i^j = 3$. This structure requires two entries in each position of the G-matrix and the P-vector. EC then performs as previously but reports each circuit $\Pi_i k_i^j$ times, where the product is over all the arcs of the circuit. Thus it is seen that the paths are "weighted" by allowing multiple arcs.

The findings on the speed of the algorithm in the previous section give a lower bound on the speed of any search algorithm performing a circuit search. The empirical evidence of Figure 4 indicates that EC would be costly to utilize on a graph with a high arc-to-vertex density and containing more than 50 arcs or 7 vertices. One would guess that this limits the practicality of application to graphs of average density to those with less than 100 arcs.

RECEIVED OCTOBER, 1969; REVISED AUGUST, 1970

REFERENCES

1. BERGE, CLAUDE. *The Theory of Graphs and its Applications*. Wiley, New York, 1966.
2. GOTLIEB, C. C., AND CORNEIL, D. G. Algorithms for finding a fundamental set of cycles for an undirected linear graph. *Comm. ACM 10*, 12 (Dec. 1967), 780–783.
3. KNUTH, D. E. *The Art of Computer Programming, Vol. 1*. Addison-Wesley, Reading, Mass., 1968.
4. PATON, KEITH. An algorithm for finding a fundamental set of cycles of a graph. *Comm. ACM 12*, 9 (Sept. 1969), 514–518.
5. ROBERTS, S. M., AND FLORES, BENITO. Systematic generation of Hamiltonian circuits. *Comm. ACM 9*, 9 (Sept. 1966), 690–694.
6. TRAIGER ,I. L., AND GILL, A. On an asymptotic optimization problem in finite, directed, weighted graphs. *Inform. Contr. 13*, 6 (Dec. 1968), 527–533.
7. WELCH, J. T. A mechanical analysis of the cyclic structure of undirected linear graphs. *J. ACM 13*, 2 (Apr. 1966), 205–210.