AN ABSTRACT OF THE THESIS OF

Jeffrey Lee DeCurtins for the degree of Master of Science

in Computer Science presented on August 12, 1977

Title: A COMPARISON OF METHODS FOR FINDING HAMILTONIAN CIRCUITS ON GRAPHS

Abstract Approved: Redacted for Privacy

(Curtis Cook)

This paper compares three classes of algorithms for finding Hamiltonian circuits in graphs. Two of the classes are exhaustive search procedures and this study finds them to have an exponential dependence on the size of the graph. The third class of algorithms, based on Warnsdorff's rule, is found to be less dependent on the size of the graph but restricted to finding only a single circuit. The effect of density and regularity of the characteristic performance of the algorithms is shown to be minimal.

A Comparison of Methods For Finding Hamiltonian Circuits in Graphs

by

Jeffrey Lee DeCurtins

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Commencement June 1978

APPROVED:

Redacted for Privacy

Associate Professor of Computer Science

Redacted for Privacy

Chairman, Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented August 12, 1977

by Jeffrey Lee DeCurtins

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

A COMPARISON OF METHODS FOR FINDING HAMILTONIAN CIRCUITS IN GRAPHS

I. INTRODUCTION

In the year 1859, Sir William Rowan Hamilton invented a puzzle and sold it to a game manufacturer in Dublin. The puzzle consisted of a regular dodecahedron, which is a polyhedron with 12 faces and 20 corners, each face a regular pentagon with three edges meeting at each corner. The corners were marked with the names of 20 important cities and the object of the game was to find a route along the edges of the dodecahedron passing through each of the 20 cities exactly once and returning to the starting point. This was the beginning of the concept of a Hamiltonian circuit.

Before generalizing the concept of a Hamiltonian circuit on a general graph, a simple graph and some of its properties shall be defined. Abstractly, a graph is a pair (V,E) where V is a finite set of objects called vertices and E, the set of edges, is a set of ordered pairs of vertices. A common representation of such a graph is shown in Figure 1. The graph is known as a digraph (directed graph) with edges having an implied direction corresponding to the ordered pairs of vertices in the set of edges.

The graph shown in Figure 1 may be represented in a number of ways. One of the most useful, the adjacency matrix, is shown in Figure 2. The adjacency matrix of a graph on N vertices is an NxN array X such that $x_{ij} = 1$ if (i,j) is an element of E and zero otherwise.

The degree of a vertex refers to the number of edges incident upon it. This may be broken down to in-degree and out-degree referring to the number of edges entering or leaving the vertex, respectively. The adjacency matrix was used to represent the graphs in programming the algorithms which were tested in this paper.

Several properties may be defined which describe the overall features of the graph. First, the density of the graph may be defined as the average out-degree of a vertex. For the graph of Figure 1 this number is two. This may also be derived from the adjacency matrix of Figure 2 by averaging the number of non-zero entries in each row.

Secondly, the regularity of a graph may be defined as the amount of deviation from the average out-degree of a vertex in the graph. The graph presented in Figure 1 is regular of degree two. From the adjacency matrix this can be seen in that there are exactly two non-zero elements in each row.

With these definitions, the concept of a Hamiltonian circuit may be generalized to the case of arbitrary graphs by defining the circuit to be a sequence of vertices $v_1, v_2, \ldots, v_N, v_1$ that includes each vertex once and such that (v_i, v_{i+1}), $1 \leq i \leq N-1$, and (v_N, v_1) are elements of E. A Hamiltonian circuit in the graph of Figure 1 is 1,3,4,5,2,1 . To date the problem of finding a necessary and sufficient graph theoretic condition for the existence of a Hamiltonian circuit in an arbitrary graph is unsolved. (1) The task of finding a Hamiltonian circuit in general might be difficult in the sense that there might be no easier way than to look at all the N! possible permutations of the

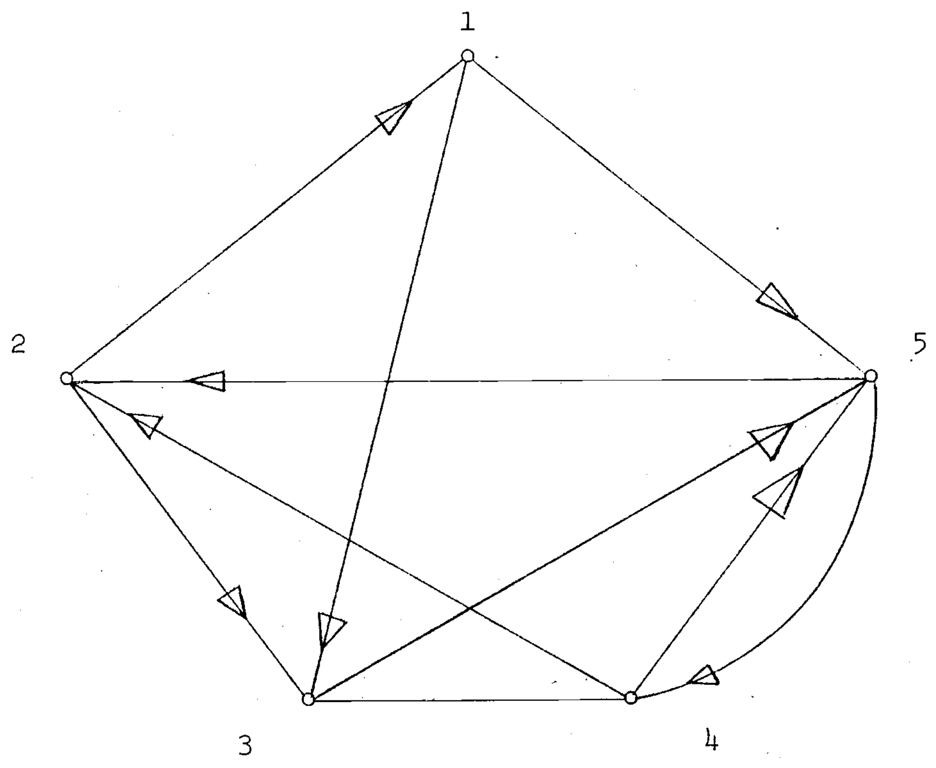

Figure 1.   A Directed Graph

$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 0 & 1 & 0 & 1 \\
2 & 1 & 0 & 1 & 0 & 0 \\
3 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 1 \\
5 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

Figure 2. Adjacency Matrix of the Graph of Figure 1

vertex set and test each for the circuit property described above. (2)

A useful structure in analyzing the search for a Hamiltonian circuit is the move tree for a graph as shown in Figure 3. The root of the tree, centered at the top of the figure, is the initial vertex from which the search is conducted. In Figure 3 the root is C. The branches of the tree are the chains of edges leading out from the root and representing all of the possible paths leading back to the initial vertex. The tree of Figure 3b shows the graph of 3a to have three such possible routes. The depth of a branch in the tree is the number of edges traversed until the initial vertex is reached again.

The purpose of this paper is to examine the various classes of algorithms which have been proposed for finding Hamiltonian circuits in arbitrary graphs and to compare their effectiveness with respect to the general properties of the graph being examined. These properties include the size of the graph (number of vertices), the density of the graph and the regularity which may be present in the graph.

II. CLASSES OF ALGORITHMS

In this chapter, the list of proposed algorithms for finding Hamiltonian circuits is broken into four classes. These classes may be labeled algebraic, permutation, deduction, and the Warnsdorff rule. The basic steps involved in the execution of each class are outlined along with examples of the procedure. The advantages and disadvantages of the various classes are also discussed.

A. ALGEBRAIC

This class of algorithms originates from early attempts at finding Hamiltonian circuits and involves the simultaneous generation of all circuits by successive matrix multiplications. (3-6) The algorithm uses the variable adjacency matrix, B, in which $B(i,j)= v_j$, the vertex label, if $X(i,j)= 1$ in the standard adjacency matrix. The possible paths, generated during execution, are symbolically stored in a column vector $A_m(i)$ of length N where N is the number of vertices in the graph. The vector is modified during the program by successive multiplications by B and successive deletions as described in the outline below.

S1: for m=1, let $A_m(i) = X(i,1)$ for i = 1 to N

S2: multiply A_m by the variable adjacency matrix B

$$A'_{m+1}(i) = \sum_k B(i,k)*A_m(k)$$

and store result in the vector A'_{m+1}

S3: for i = 1 to i = N, delete all terms of the resulting $A'_{m+1}(i)$ containing vertex label v_i and after deletion set m = m+1 and rename A'_{m+1} as A_m

S4: if m is less than N, go to step S2

S5: if m = N then element $A_N(1)$ contains a symbolic list of all Hamiltonian circuits in the graph

This class of algorithm suffers from a critical drawback. The amount of storage space required to symbolically keep track of all the partial paths generated after each step quickly becomes prohibitive. As will be seen in the next section, for some graphs the number of partial paths near the end of execution may be on the order of $N!$. Results obtained elsewhere show this method to be much less efficient than some of the other classes of algorithms examined below and therefore the algebraic class will not be discussed further. (7)

B. PERMUTATION

This class of algorithms employs a straightforward systematic and exhaustive search. (8) The basic procedure includes the systematic consideration of every possible path from the first or initial vertex.

The descriptive outline presented below is characteristic of the procedure followed by an algorithm of this class. The initial vertex is taken as vertex 1 and all branching possibilities from this vertex are considered.

```
S1: start with vertex 1 as current vertex
S2: find next branch possibility from current vertex
    to a vertex not in the partial path
S3: if a branch possibility exists, advance partial
    path to this vertex, make this vertex current and
    go to step S2.
S4: if no further branch possibilities exist and the
    current vertex is vertex 1, no more circuits exist,
    END.
S5: if no branch possibilities exist, backtrack to the
    last vertex from which a branch occurred, goto S2.
S6: if all vertices are in partial path and next branch
    is to vertex 1 then a Hamiltonian circuit exists;
    record the circuit and go to step S5.
```

The algorithm has the advantage of being easy to implement and conceptually straightforward. The procedure is an exhaustive search and thus will eventually find all existing Hamiltonian circuits in the graph. It is a brute force method which steps through all possibilities, the number of which could become quite large for a very dense graph. The amount of time taken will depend largely on how quickly the possible paths terminate.

In Figure 3 a small graph with N = 4 vertices and the tree representing the possible partial paths to the initial vertex is given. Table 1 gives the sequence of moves which a permutation algorithm would follow in traversing the tree. The algorithm searches every branch of the move tree to its end and, in this case takes nine steps to complete the process.

The graph shown as an example in Figure 3 has few edges incident on each vertex. If the right branch of the tree (C-B-D-A-C) were labeled differently so that it was the first branch to be searched, the algorithm would have taken four steps to find a circuit, that is, order(N) steps where N is the number of vertices in the graph. This is the minimum number of steps required for a complete circuit since the search must be carried to a depth of N in the move tree.

In a graph of N vertices in which every vertex is connected by an edge to every other vertex, that is, N-1 edges incident on each vertex, it can be seen that at a depth of k in the move tree, each vertex will have (N-1-k) possible branches. For each of these branch possibilities at depth k there will be a vertex at depth k+1 with (N-1-(k+1)) branch possibilities. Letting B_k represent the number of branches at depth k gives:

$$B_{k+1} = B_k\,(N-1-(k+1)) \qquad B_o = (N-1)$$

Starting with $k = 0$ and continuing through the series to $k + 1 = N$ gives:

$$B_{k+1} = B_o \,(N-1-(1))(N-1-(2))(N-1-(3)).....(N-1-(N-2))$$

or:

$$B_N = (N-1)!$$

Thus in the worst case, the permutation algorithm might search order((N-1)!) possible paths.

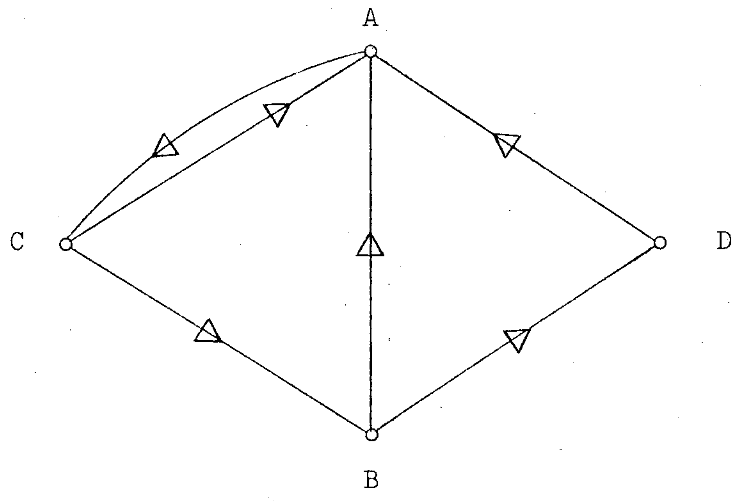

Figure 3a. Graph on Four Vertices

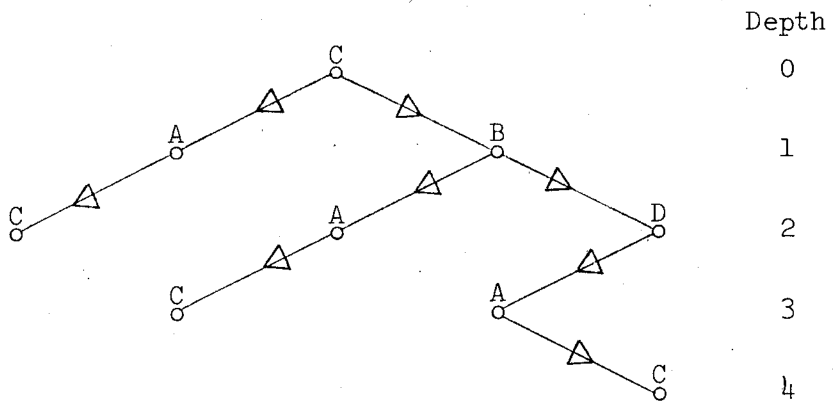

Figure 3b. Move Tree of Graph in 3a

TABLE 1

Permutation Algorithm on the Graph of Figure 3a

Step #	Partial Path	Comments
1	C	
2	C-A	
3	C-A-C	fail: reaches initial vertex too soon, back-track to vertex C
4	C-B	
5	C-B-A	
6	C-B-A-C	fail: backtrack to vertex B
7	C-B-D	
8	C-B-D-A	
9	C-B-D-A-C	circuit

C. DEDUCTION

The algorithms in this class consist of the same basic procedure as the permutation class, that is, exhaustive search of all possibilities for branching. In this class, however, deduction rules are used at each step to eliminate as many branches as possible. (9,10) The branches eliminated are those that will lead to a dead end path.

In the descriptive outline given below, the initial vertex is again taken as vertex 1 and all branch possibilities are considered which cannot be eliminated by the deduction rules. In step S2, an admissible partial path is one to which a failure rule does not apply. These rules are also given below.

S1: select vertex 1 as the initial path

S2: test the path for admissibility deleting all branching possibilities which can be shown to lead to dead ends.

S3: if partial path is admissible, find next possible branch vertex and extend path to this vertex, goto step S2.

S4: if partial path is inadmissible, delete from the partial path the last vertex to which a branch occurred, find the next possible branch from vertex at end of partial path and extend path to this vertex, goto step S2.

S5: if all branch possibilities from a given vertex have been shown inadmissible, goto step S4.

S6: if all branches from vertex 1 have been examined, no further circuits exist, END.

S7: if all vertices are in partial path and next branching possibility is vertex 1 then a Hamiltonian circuit exists; record the circuit and goto step S4.

The test for admissibility in S2 is made by applying the deduction rules below. Each time the test is applied, some edges may be deleted and some may be made required edges (those which must be present in the circuit if it is to contain the current partial path).

R1: if a vertex has only one edge entering(leaving) then that edge is required.

R2: if a vertex has a required edge entering (leaving) then all other edges entering (leaving) may be deleted.

R3: fail if any vertex has no edge entering (leaving).

R4: fail if any vertex has two or more required edges entering (leaving).

Whenever a fail applies, the partial path becomes inadmissible and backtracking must occur.

This class of algorithm is much more difficult to implement than the permutation class described above. This is due to the rules which must be incorporated and the bookkeeping measures needed to classify edges and backtrack from dead ends. The search time, however, should be less than the permutation method due to the reduction in the number of branching possibilities at each step since the elimination of some partial paths by the rules results in a smaller average length of partial path explored before reaching a dead end. This may be seen in the example of Table 2, which lists the steps taken by the deduction algorithm in traversing the move tree of the graph in Figure 3. Whereas the permutation algorithm takes nine steps to complete a circuit, the deduction method, by the elimination of some of the dead end branches, uses only six steps.

TABLE 2

Deduction Algorithm on Graph of Figure 3a

Step #	Partial Path	Comments
1	C	
2	C-A	eliminate BA,DA; fail by R3 since D has no edge leaving; backtrack to vertex C
3	C-B	edge BA is eliminated by R2 since BD is required
4	C-B-D	
5	C-B-D-A	
6	C-B-D-A-C	circuit

The advantages to this algorithm may be outweighed by the extra time required to apply the deduction rules at each step. The application of these rules may require more time than checking all possible partial paths. The application of each rule is quadratic in nature since all of the N^2 elements of the adjacency matrix must be examined. If the straightforward permutation algorithm for a particular graph is exponential in time then the application of the deduction rules may provide for a reduction in the processing time.

D. WARNSDORFF

An algorithm of this class is centered around the application of a heuristic procedure which may be viewed as an approximation to a full search of the tree representing the possible paths from an initial vertex.

The rule was originally proposed by H. Warnsdorff in 1823 for finding knight's tours on a chess board. A knight's tour is a Hamiltonian circuit on the graph formed by using the squares as vertices and a knight's possible moves as edges between squares. Warnsdorff's rule, as originally proposed, called for the selection of the next move of the knight based on a calculation of which move connected with the fewest number of further moves. If a tie occurs it may be broken arbitrarily. Thus for a knight in the middle of a chess board there are moves to eight possible squares and Warnsdorff's rule requires that the number of possible moves from each of these eight squares be found and the knight moved to the square with the least number of further moves.

This rule has been generalized to higher order calculations by Pohl. (11) The generalized Warnsdorff's rule considers all paths of k moves and counts the remaining number of connections for each path. The path with the minimum number of connections at the k^{th} move is the one who's first move is selected, provided that this number is not zero. Ties are broken by going to k + 1 moves. Thus for Warnsdorff' rule as originally stated, k = 1.

Pohl's suggested generalization of Warnsdorff's rule requires the summation of the degrees of vertices in each possible partial path of length k from the current vertex. It should be noted that this is not the only possible generalization of Warnsdorff's rule. An alternative to the summation might be the selection of the vertex which connects with a further vertex of minimum degree. Other possibilities might include some form of combination of the above alternatives.

Since Warnsdorff's rule is only an approximation to a full search of the move tree, cases exist for which it will not work, that is, it will not find a complete Hamiltonian circuit even if one is present. It is then necessary to complete the search for a circuit by another algorithm as described below:

```
S1: pick initial vertex as current vertex, goto S5
S2: find successor of current vertex with least number of
    branches greater than zero, add to partial path and
    make it the current vertex
S3: counting branches to the initial vertex, if the degree
    of any vertex not in partial path is zero, go to the
```

```
        alternate algorithm
S4:     if partial path includes all vertices and branch from
        current vertex to initial vertex exists then DONE
S5:     reduce by one the degree of any vertex connected to
        the current vertex and go to step S2
```

Warnsdorff's rule and the algorithms based upon it suffer from the disadvantage that the method is not guaranteed to work. It will always work if taken to a high enough order but to do so would become costly since this operation would become equivalent to the permutation method of searching the full move tree. However, if the rule reaches a deadlock, it may be possible to find a circuit by continuing the search using a second algorithm to handle the remaining portion of the graph. The results of this paper were obtained by appending the permutation algorithm to the processing performed by Warnsdorff's rule.

Warnsdorff's algorithm also has the disadvantage that it is not and exhaustive search. The other classes of algorithms will systematically find all circuits but a Warnsdorff algorithm of an order less than the number of vertices is only a partial search of the move tree and thus will not find all circuits. The algorithm may be useful in cases where only a single circuit is desired.

The implementation of Warnsdorff algorithm is not straightforward. The higher order tie-breaking procedures are difficult because of the rapidly expanding number of partial paths as the order increases. Also, in appending a second algorithm for use when

Warnsdorff's rule fails, it may be difficult to match the initial conditions for starting the second algorithm. For example, the deduction class has an elaborate set of bookkeeping procedures which accrue as the program is executed. It may be difficult to jump into the middle of processing from a Warnsdorff algorithm which has not been using those bookkeeping procedures.

Warnsdorff's rule has the advantage of being fast. For first order searching, it requires only order(N) tests to advance the partial path. The second or higher orders of tests will be necessary only in case of two or more branch possibilities of equal degree, which is likely only if the graph is highly regular. The operation of Warnsdorff's rule can be seen in Table 3. Only five steps are required where the deduction algorithm needs six and the permutation nine. In a Warnsdorff algorithm, the sooner the rule reaches a deadlock, the more the performance will approach that of the appended algorithm.

TABLE 3

Warnsdorff Algorithm on

Graph of Figure 3a

Step #	Partial Path	Comments
1	C	B has connectivity of two, A has connectivity of zero because it is connected to the initial vertex which is current and so has had its degree reduced
2	C-B	A is again excluded so go to D
3	C-B-D	the only edge is to A
4	C-B-D-A	
5	C-B-D-A-C	circuit

III. EXPERIMENTAL DESIGN AND RESULTS

A. DESIGN

The experiment was designed for the comparison of the algorithms in two categories: performance in finding a single circuit in the graph and performance in finding all the circuits of a graph. In finding a single circuit, the algorithms were tested on graphs with several specific properties to determine if variation of these properties effected the performance of the algorithm. These properties include the size of the graph, density and the regularity.

The data used in the experiment consisted of random graphs generated by a routine which allowed for variation of the size, density and regularity of the graph. The basic design of the graph generation program randomly placed 1's in the rows of an array which became the adjacency matrix for the graph. The size of the graph was controled by the dimension of the array. The density of the generated graph was dependent upon the number of non-zero elements placed in each row of the array. For a given density, each row of the array must be filled with say k (k = density) ones placed randomly into N-1 possible locations, the diagonal elements of the array being excluded to eliminate self-loops. This was accomplished by forming the list of integers from one to N, scrambling the list using a random number generator and using the first k entries of the scrambled list as indices for ones in the current row of **the adjacency** matrix. The regularity was adjusted so that each vertex had a possible range of degree from k-r to k+r where r is an integer less than or equal to k. This was done by randomly choosing a number between -r and +r and adding it to k. The new value of k was then used in the degree procedure above controlling density. The entire process was repeated for each row of the adjacency matrix. The values of N,k and r were the input parameters to the graph generation routine.

To insure the presence of a Hamiltonian circuit, the graph generation program superimposed upon the already existing adjacency matrix, a second adjacency matrix consisting simply of a single Hamiltonian circuit. This circuits was made random by first generating an array containing the simple circuit 1,2,3,...,N,1 and then using a scrambled list of the integers from one to N as a mapping to rename the vertices of the simple circuit. When this Hamiltonian circuit was added to the original adjacency matrix, at most one non-zero element was added to each row thus making the average density somewhere between k and k+1. Therefore most of the graphs generated are not perfectly regular but only approximately regular.

Regression analysis and analysis of variance were used to form models of the behavior of the algorithms and to place limits on the reliability of those models. Many authors do not study the reliability of the results for performance of algorithms. Instead, the usual procedure is to report the execution times for various test runs of the algorithm on a particular machine. In this paper, rather than giving the time of execution as a measure of performance, the number of operations necessary to reach the completion of of the program were used as a measure of performance. To do this, a counter was placed in the programs and the number of arithmetic and memory operations was recorded. Arithmetic operation included the operations of addition, subtraction, multiplication,division and assignment. Memory operations consisted of array references. A combination of the two indicators was then used for the computational complexity of the program and thus, the algorithm. The final measurement variable consisted of the weighted sum of the operations classed as arithmetic and memory with the latter having two times the weight of the former. The analysis of performance was then obtained using three repetitions per treatment, that is, three repetitions for every combination of the parameters.

A major goal of the experiment was to implement and compare the behavior of Warnsdorff's algorithm with the other two since no previous comparative studies had ever included a Warnsdorff algorithm. Also, the characteristics of the Warnsdorff algorithm in conjunction with the hybrid formulation described above needed study.

In the second part of the experiment, the permutation and deduction algorithms were compared for finding all the circuits of a graph. The average amount of computation per circuit was used as the measure thus making it independent of the number of circuits in a particular graph. In this portion of the experiment the primary interest was the dependence of the algorithms on the size of the graph, the number of vertices, N, and how that compares with the N dependence of the algorithm in finding only a single circuit.

B. RESULTS

The general N dependence of the algorithms in finding a single circuit is displayed in Figure 4. This figure shows the log of the number of operations versus the size of the graph. The fitted regression lines for the permutation and deduction algorithms are given along with some of the data points for each of the three algorithms.

The preliminary attempts to fit regression models to the three algorithms yielded a possible exponential dependednce for both the permutation and deduction algorithms but only an N or N^2 dependence for the Warnsdorff algorithm. As can be seen from Figure 4, performance of the Warnsdorff algorithm was not consistent and a high degree of fit with a regression model could not be obtained. Thus the use of a two-factor analysis of variance was needed to decide if a significant difference did exist between Warnsdorff and the permutation or deduction groups. The two-factor analysis of variance given in Table 4 concluded that there was a significant difference between the N dependence of the permutation and Warnsdorff groups. This can be seen from the large F value for the interaction mean square. The minimum F value for a 95% confidence level signifying a difference is approximately 8.70 and the calculated F value was 11.05 . Thus it can be concluded that the two algorithms differ in their dependence on the size of the graph.

Table 5 contains the results from regression analysis on the

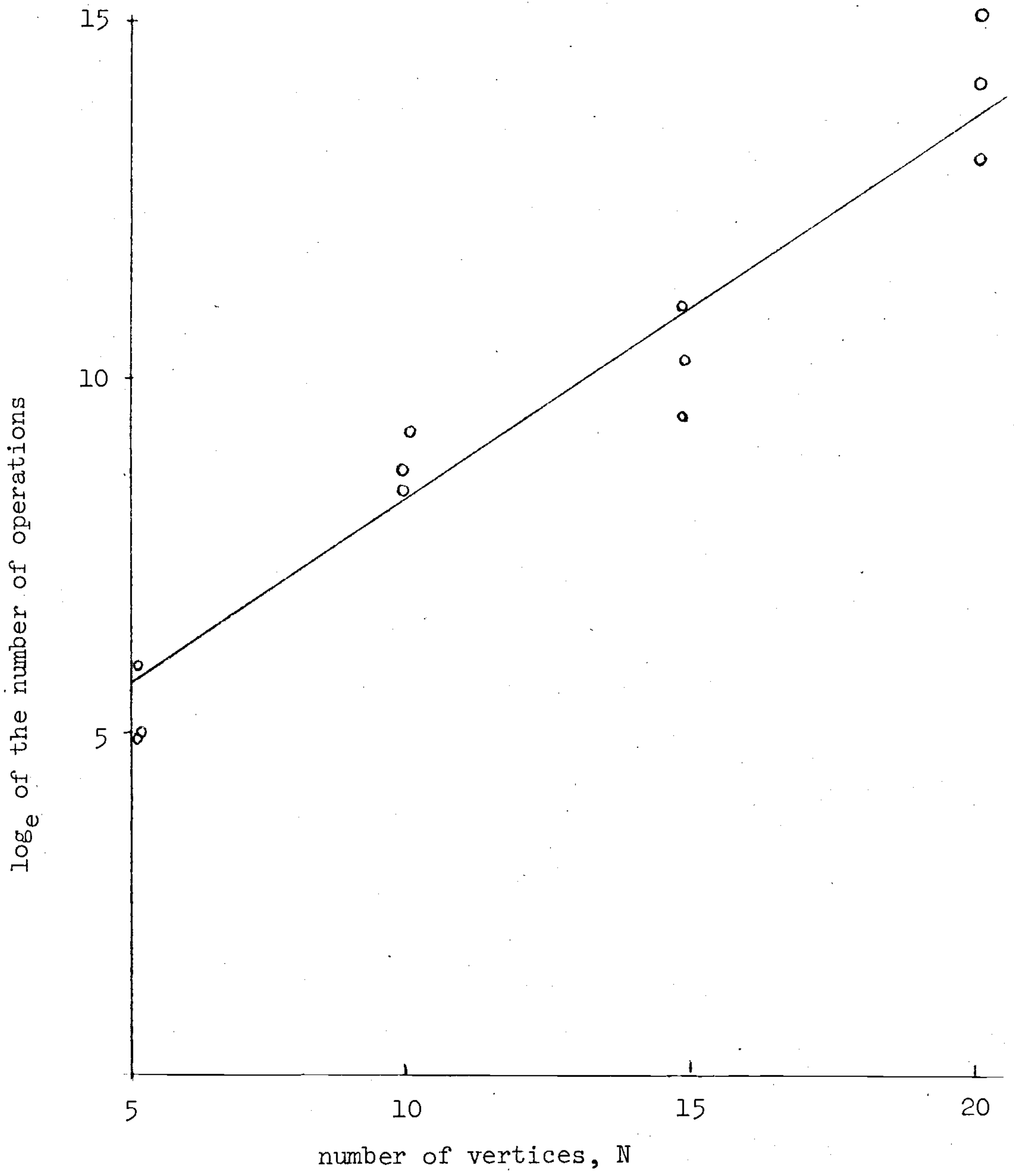

Figure 4. REGRESSION FIT OF DATA FOR PERMUTATION PERFORMANCE ON RANDOM GRAPHS (FIRST CIRCUIT)

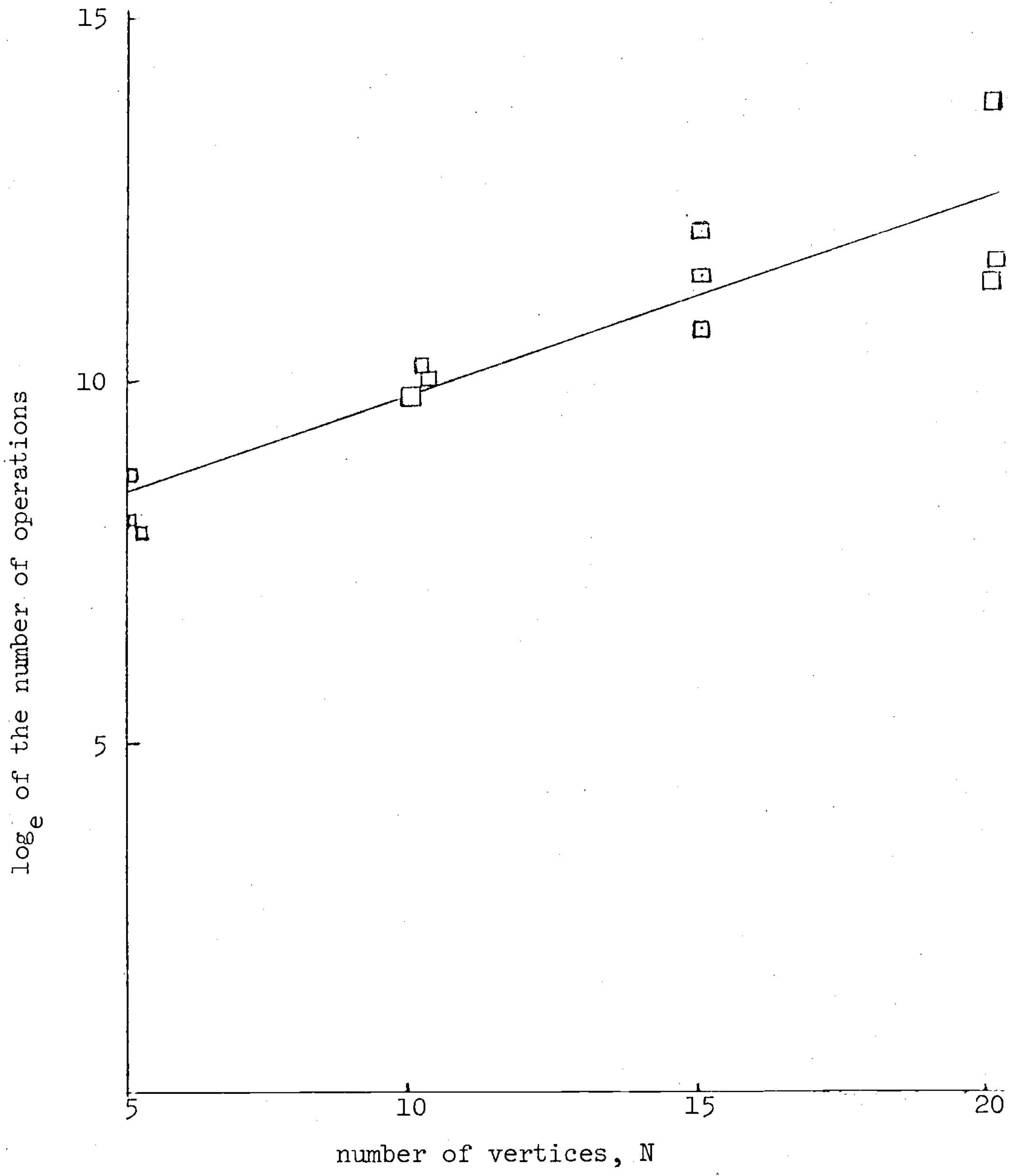

Figure 4b. REGRESSION FIT OF DATA FOR DEDUCTION PERFORMANCE ON RANDOM GRAPHS (FIRST CIRCUIT)

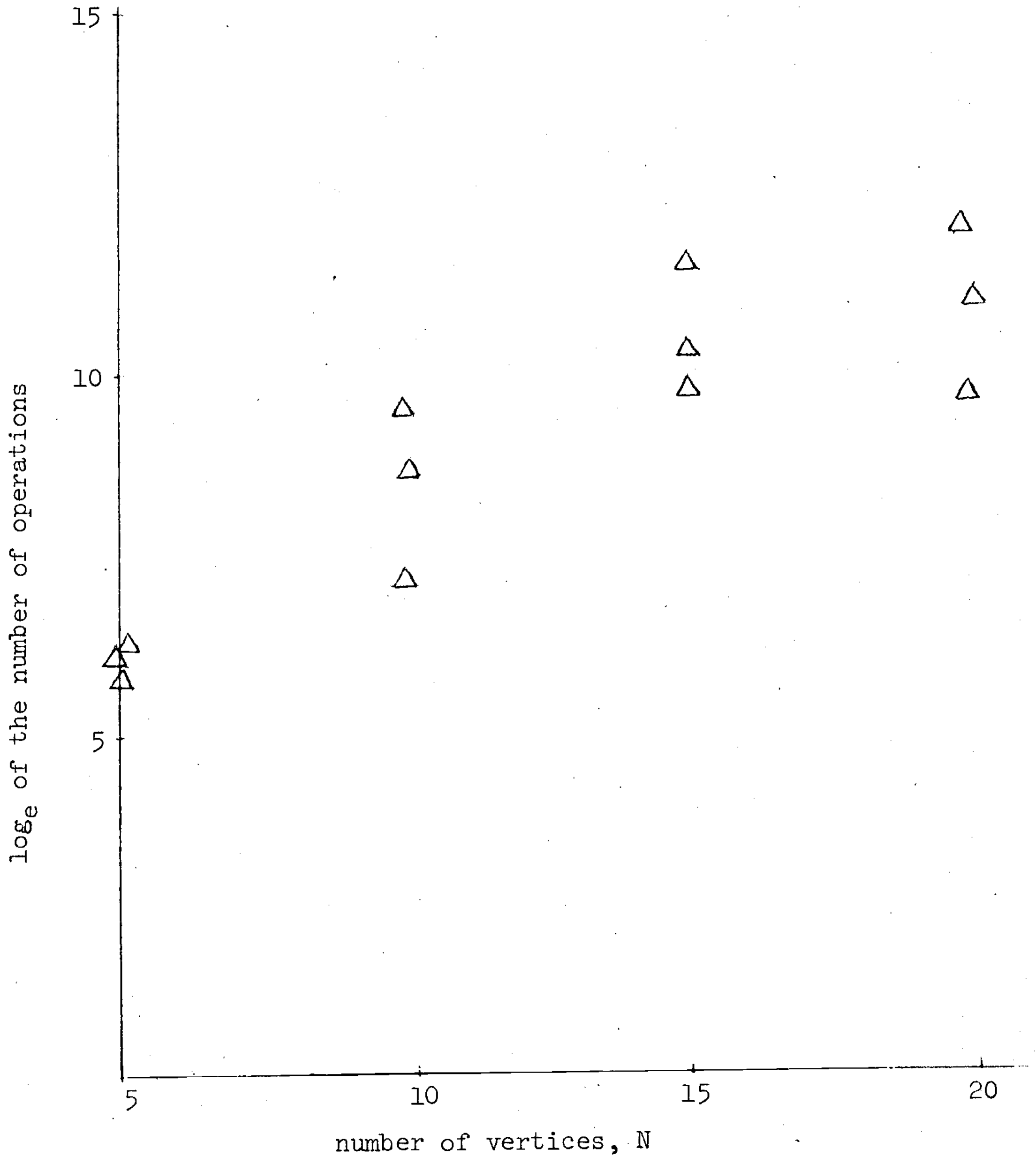

Figure 4c. DATA POINTS FOR WARNSDORFF PERFORMANCE ON RANDOM GRAPHS (FIRST CIRCUIT)

TABLE 4. TWO FACTOR ANOVA

(Warnsdorff,permutation)vs(N)

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Square	F
TOTAL	1.14×10^{13}	23		
TREATMENT	7.15×10^{12}	7	1.02×10^{12}	
Algorithm	9.14×10^{11}	1	9.14×10^{11}	3.47
N	3.33×10^{12}	3	1.11×10^{12}	4.22
Interaction	2.91×10^{12}	3	9.69×10^{11}	11.05
ERROR	4.21×10^{12}	16	2.63×10^{11}	

performance of the permutation and deduction algorithms. The analysis of variance shows a high degree of fit (91.5%) on the transformed data (the $\log_e$ of the number of operations) and the set of t values for the parameters N, PERM and N*PERM all show a significance above the 95% confidence level.

The fitted models are then:

$$\text{OPERATION}_{\text{perm}} = 17.5 \exp(.540N)$$

$$\text{OPERATION}_{\text{ded}} = 1108 \exp(.274N)$$

The above equations and their graphs in Figure 4 show clearly that the deduction algorithm is less efficient than the straightforward permutation method for N smaller than approximately 16. This can be attributed to the extra processing necessary to apply the deduction rules. But with an increasing number of vertices, the exponential nature predominates and the deduction algorithm, with smaller exponent, prevails.

For Warnsdorff's algorithm in Figure 4, only the data points are plotted but the advantage is clear. The points do not indicate a straight line on the log plot and thus show Warnsdorff's procedure is not of an exponential nature. It should be reiterated that the algorithm is basically linearly dependent on N until it fails. Failure or a resort to higher orders of the rule is most likely only for highly regular graphs.

Table 6 contains the two-factor analysis of variance to determine if the character of various algorithms change with a change in the density of the graph. The table does show

TABLE 5. REGRESSION AND ANOVA
FOR PERMUTATION AND
DEDUCTION (first circuit)

Transformed Model:

$$\log_e(\text{OPERATIONS}) = 7.02 + .274\,N - 4.16\,\text{PERM} + .266\,N{*}\text{PERM}$$

<u>t values</u>:

Variable	Std. Error	t
CONSTANT	.574	12.2
N	.042	6.54
PERM	.811	5.13
N*PERM	.059	4.50

<u>ANOVA</u>:

Source	df	SS	MS
TOTAL	23	155	6.75
REGRESSION	3	141	47.3
RESIDUAL	20	13.1	.658

$R^2 = .915$

Model:

$$\text{OPERATIONS}_p = 17.5 \exp(.540N)$$

$$\text{OPERATIONS}_d = 1108 \exp(.274N)$$

a difference in the performance of the algorithms on graphs of differing densities, with the computed F at 11.32 indicating a confidence level for the effect of better than 90%. This is to be expected since there are more partial paths to be explored. However, there is no apparent change in the characteristic performance of the algorithms with changing density. This is evidenced by the very small interaction term, F= .22, shown in Table 6.

Table 7 contains the analysis of variance and regression model for interpreting the effect of regularity on the permutation algorithm. The two models providing performance as a function of N are:

$$\text{High Regularity:} \quad \text{OPERATIONS}_h = 23.7 \exp(.496N)$$

$$\text{Low Regularity:} \quad \text{OPERATIONS}_l = 27.5 \exp(.507N)$$

The equations are essentially identical indicating that regularity of a graph has virtually no effect on the performance of the permutation method. This is further evidenced by the t values for the parameters which differentiate between data for regular and non-regular graphs. For a 60% significance, the minimum t value is .941 and both parameters REG and N*REG have t values far below this level.

Table 8 contains similar results for the parameters which differentiate regularity in the test on the deduction algorithm. There too, the t values for those parameters fail to reach even the 60% confidence level. Thus again, as with the permutation method, regularity of the graph appears to have no effect on the perform-

TABLE 6. TWO FACTOR ANOVA

(density)vs(algorithm)

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	F
TOTAL	17	8.84×10^{10}		
TREATMENT	5	4.76×10^{10}		
Density	1	3.85×10^{10}	3.85×10^{10}	11.32
Algorithm	2	7.54×10^{9}	3.77×10^{9}	1.11
Interaction	2	1.51×10^{9}	7.57×10^{8}	.22
ERROR	12	4.09×10^{10}	3.40×10^{9}	

TABLE 7. REGRESSION AND ANOVA FOR PERMUTATION ON REGULAR VS NON-REGULAR

Model (transformed):

$$\log_e (\text{OPERATIONS}) = 3.32 + .507\,N - .150\,\text{REG} - .011\,N{*}\text{REG}$$

t values:

Variable	Std. Error	t
CONSTANT	.984	3.37
N	.072	7.06
REG	1.39	.108
N*REG	.102	.110

ANOVA:

Source	df	SS	MS
TOTAL	7	65.7	9.39
REGRESSION	3	63.1	21.0
RESIDUAL	3	2.58	.645

$R^2 = .961$

Model:

$$\text{OPERATIONS}_p = 27.5 \exp(.507N)$$

$$\text{OPERATIONS}_{p.r} = 23.7 \exp(.496N)$$

ance of the algorithm.

Table 9, a two-factor analysis of variance, shows at greater than the 60% confidence level the regularity of a graph does have an effect upon the performance of the Warnsdorff algorithm. Given the method of Warnsdorff's rule, comparing the number of edges incident on adjacent vertices and going to higher order in case of a tie, the regularity of a graph might be expected to have a greater effect upon the performance of this algorithm. However, the graphs used in obtaining performance data in the regular category were only approximately regular not precisely regular and thus the full effect of regularity was not measured.

Table 9 also shows an interaction, significant to greater than 90%, between the size of the graph, N, and the degree of regularity. Thus for larger graphs, the deteriorating effect of regularity on performance of the method is more pronounced. This is expected since with increasing N the number of times a tie will be encountered increases.

It is interesting to note that even for approximately regular graphs, the Warnsdorff algorithm, on the average, outperforms both the permutation and deduction methods.

In part two of the experiment, the performance of the two exhaustive search algorithms was analyzied with data representing operations needed to find all the circuits of a graph. Figure 5 shows the regression model plots of performance of the two methods. Table 10 contains the analysis of variance and regression model results pertaining to the performance of the algorithms in finding

TABLE 8. REGRESSION AND ANOVA
FOR DEDUCTION ON REGULAR
VS NON-REGULAR

Model (transformed):

$$\log_e(\text{OPERATIONS}) = 6.61 + .331\,N + .825\,\text{REG} - .085\,N{*}\text{REG}$$

t values:

Variable	Std. Error	t
CONSTANT	1.11	5.94
N	.081	4.07
REG	1.57	.524
N*REG	.115	.741

ANOVA:

Source	df	SS	MS
TOTAL	7	24.7	3.52
REGRESSION	3	21.36	7.12
RESIDUAL	4	3.30	.825

$R^2 = .866$

Model:

$$\text{OPERATIONS}_d = 742 \exp(.331N)$$

$$\text{OPERATIONS}_{d.r} = 325 \exp(.246N)$$

TABLE 9. TWO FACTOR ANOVA

(regularity)vs(N)

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Square	F
TOTAL	23	$1.63\text{x}10^{11}$		
TREATMENT	7	$1.34\text{x}10^{11}$		
Regularity	1	$1.32\text{x}10^{10}$	$1.32\text{x}10^{10}$	7.65
N	3	$7.24\text{x}10^{10}$	$2.41\text{x}10^{10}$	13.68
N*Reg	3	$4.84\text{x}10^{10}$	$1.61\text{x}10^{10}$	9.14
ERROR	16	$2.82\text{x}10^{10}$	$1.76\text{x}10^{9}$	

all circuits. The measure of performance was the average number of operations per circuit in the graph. The overall computational complexity for the permutation process is given as:

$$\text{OPERATIONS}_p = 28.2 \exp(.466N)$$

and for the deduction process as:

$$\text{OPERATIONS}_d = 1017 \exp(.283N)$$

To test for equivalence between overall performance and first circuit performance the Bonferroni joint confidence intervals were employed. To use a 94% family confidence interval, the coefficient needed is:

$$B = t(.99, 20) = 2.528$$

and the confidence intervals become, for first circuit data:

CONSTANT	7.02 ± 1.47
N	$.274 \pm .108$
PERM	4.16 ± 2.08
N*PERM	$.266 \pm .152$

and for total performance:

CONSTANT	$6.93 \pm .862$
N	$.283 \pm .063$
PERM	3.59 ± 1.27
N*PERM	$.183 \pm .084$

As is evidenced by the limits of the confidence intervals, each set of parameter values is contained within the other's confidence

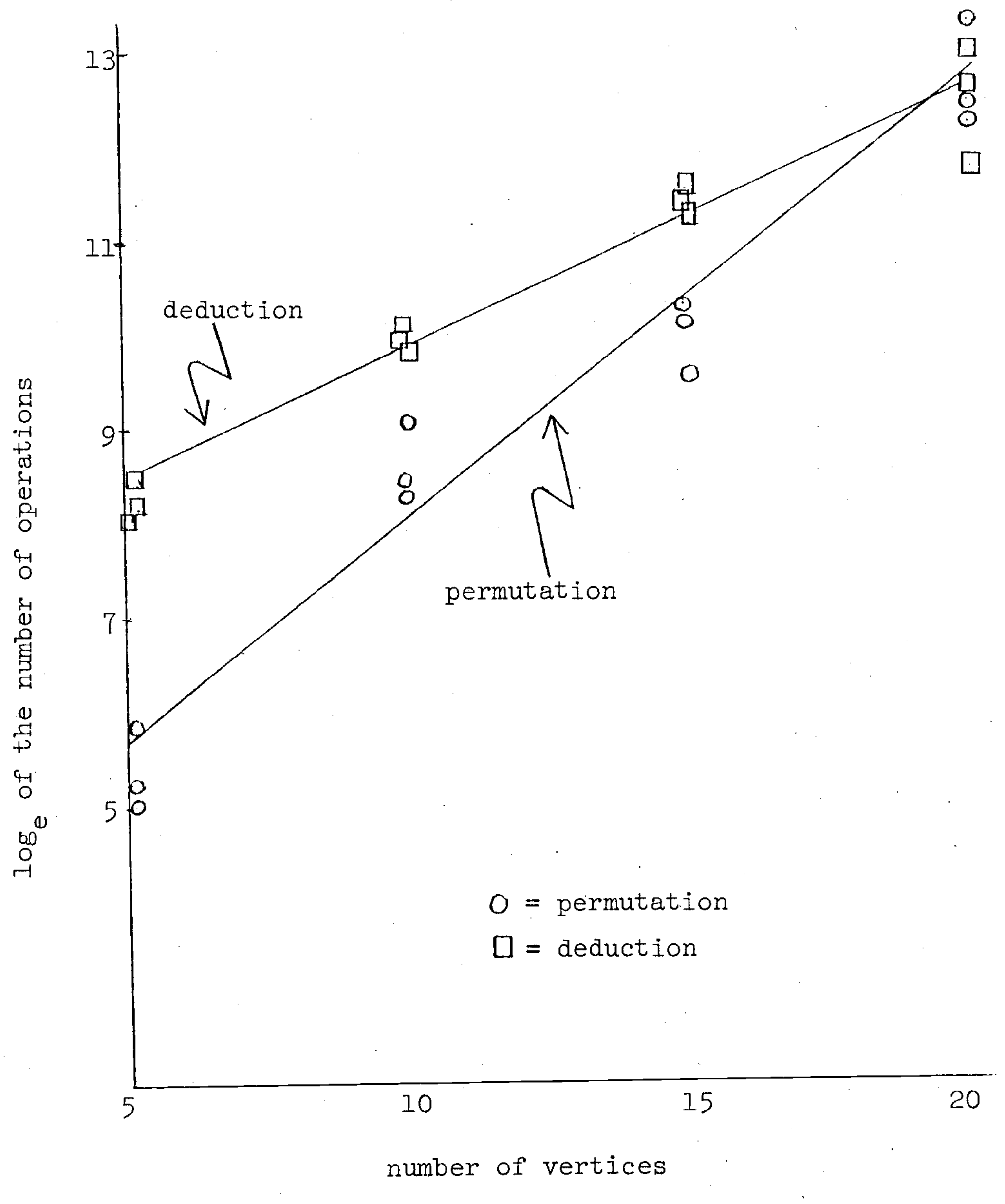

Figure 5. REGRESSION FIT OF DATA FOR ALGORITHMS ON RANDOM GRAPHS (average number of operations per circuit)

TABLE 10. REGRESSION AND ANOVA
FOR PERMUTATION AND DEDUCTION
ON ALL CIRCUITS

Transformed model:

$$\log_e (\text{OPERATIONS}) = 6.93 + .283\ N - 3.59\ \text{PERM} + .183\ N{*}\text{PERM}$$

<u>t values</u>:

Variable	Std. Error	t
CONSTANT	.341	20.3
N	.025	11.4
PERM	.482	7.44
N*PERM	.035	5.18

<u>ANOVA</u>:

Source	df	SS	MS
TOTAL	23	126	5.49
REGRESSION	3	122	40.5
RESIDUAL	20	4.65	.233

$R^2 = .963$

Model:

$$\text{OPERATIONS}_p = 28.2 \exp(.466N)$$

$$\text{OPERATIONS}_d = 1020 \exp(.283N)$$

interval. Thus it can be concluded that the performance of the algorithms is, on the average, the same for each circuit in a graph, whether the purpose is to find one circuit or all circuits. Therefore in general, for large N, the deduction algorithm's elimination of certain paths allows it to outperform the basic permutation method.

IV. CONCLUSIONS

The main concern in testing these algorithms was their dependence on the size of the graph. Both the permutation and deduction methods were seen to have an exponential dependence on N, the size of the graph. Other factors such as regularity and density were seen to have little or no effect on the overall N dependence of these two algorithms.

As expected, the application of the deduction rules called for an additional amount of computation over and above the underlying permutation structure. Thus the much simpler and more straightforward permutation algorithm outperformed the deduction for smaller graphs (experimentally, N less than 16). The application of the deduction rules clearly reduces the exponential factor in the execution time and thus, unless a very simple application on a small graph is planned, the deduction algorithm is to be preferred.

Warnsdorff's algorithm is seen to have a non-consistent behavior but is much less strongly dependent on N than either of the other two procedures. This allows for considerable savings in computational effort. Thus for very large graphs of approximately 20 or more vertices, in which only a single or a few circuits are desired, the use of a Warnsdorff algorithm is imperative.

But, as described above, a Warnsdorff algorithm suffers from two serious handicaps. First, it is not guaranteed to work and hence must be augmented by a second algorithm of the permutation or deduction type. This is an implementational not a computational drawback since in the limit of immediate failure of the Warnsdorff algorithm, the execution time simply rises to that of the appended algorithm.

A second disadvantage of the method is that it is not an exhaustive search and thus cannot be guaranteed to find all the circuits in a graph although the algorithm can easily be ammended to find possibly more than one.

Several closing comments on the testing of algorithms might be appropriate here. First, the literature displays a wide variation in the method of analysis of test results on algorithms. One author may give best and worst observed execution times on a particular machine, another the average results for a few tests on a small, restricted set of data. Some form of standardization of testing procedures and analysis is needed so published results on related algorithms might be compared more readily.

Lastly, the experimental result given in any study of the performance of algorithms is probably not independent of the quality of the programming used to implement a procedure. Thus some element of skepticism must be associated with any form of analysis where programming technique is a factor.

BIBLIOGRAPHY

1. Graph Theory with Applications to Engineering and Computer Science, N. Deo, Prentice-Hall, 1974, p30

2. The Complexity of Theorem-Proving Procedures, S. Cook, Proceedings 3rd ACM Symposium on Theory of Computing,1971, p151

3. Self-Avoiding Paths and the Adjacency Matrix of a Graph, J. Ponstein, Journal of S.I.A.M., May 1966, p600

4. Generation of all Hamiltonian Circuits Paths and Centers of a Graph and Related Problems, S. Yau, IEEE Transactions on Circuit Theory, 1967, p79

5. A Systematic Method of Finding All Directed Circuits and Enumerating All Directed Paths, T. Kamae, IEEE Transactions on Circuit Theory, 1967, p166

6. On Finding the Simple Paths and Circuits in a Graph, G. Danielson, IEEE Transactions on Circuit Theory, 1968, p294

7. Graph Theory: An Algorithmic Approach, N. Christofides, Academic Press, 1975, p217

8. Systematic Generation of Hamiltonian Circuits, S. Roberts and B. Flores, Communications of the ACM, Sept. 1966, p690

9. A Search Procedure for Hamiltonian Paths and Circuits, F. Rubin, Journal of the ACM, Oct. 1974, p576

10. Recent Progress and New Problems in Applied Graph Theory, S. Hakimi, IEEE Region Six Conference Record, 1966, p635

11. A Method for Finding Hamiltonian Paths and Knight's Tours, I. Pohl, Communications of the ACM, 1967, p446