Technical Report CS82012

Performance Evaluation of Three Algorithms to Generate Fundamental Cycles for Twelve Different Graphs

December, 1982

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

1.0 INTRODUCTION

Current research in algorithms to generate fundamental cycles in a graph is being reported in Deo et al. [1], Egyhazy [2] and Paton [3]. Both are focused on developing and measuring the performance of heuristic algorithms to solve the problem while minimizing the computational complexity. The computational complexity of algorithms to generate a set of fundamental cycles is of the order of the total length L of the fundamental cycle set with respect to a spanning tree T in graph G. Since the total length is dependent on T, we are interested in obtaining a spanning tree for which $L(T) - L(T_{min})$ is small.

The objective of the research effort reported in this paper is an evaluation of the time complexity of three algorithms, BCG [3), SDS [1] and MBFSUE [2], for a total of 12 different graphs. Each algorithm is implemented using the three most common graph representations, namely the adjacency matrix, one way adjacency lists and adjacency multilists. In general, we are persuing the determination of two major dependent variables, the average length of the set of fundamental cycles found by each algorithm and the CPU minutes consumed by the implementation of each algorithm-graph representation combination, for 12 intuitively interesting graphs.

2.0 THE THREE HEURISTIC ALGORITHMS

2.1 BACKGROUND

The Basic Cycle Algorithm (BCG) was formulated by Paton [3] as a fast method for finding a fundamental set of cycles. A spanning tree is first generated and the vertices examined in turn. The unexamined vertices are stored in a pushdown list awaiting examination. The second step is to take the top element t of the pushdown list and examine it; i.e., inspect all those edges (t,z) of the graph for which z has not yet been examined. If z is already in the tree, a fundamental cycle is added; if not, the edge (t,z) is placed in the tree. The algorithm terminates when all the unexamined vertices have been examined. In general, BCG is classified among algorithms that employ a mixed search method in traversing the graph.

The Static Degree Sort (SDS) algorithm, formulated by Deo [1], generates the spanning tree in a straightforward breadth first fashion. The first vertex explored from is the vertex with highest degree. The next vertex considered is the one with the highest degree among the successors of the oldest vertex explored from and not necessarily the vertex with the highest degree among those that are already present in the partial tree.

The Multipoint Breadth First Search with respect to "unexplored" edges conceived by Egyhazy [2], extends Deo's [2] original MBFS, which says that we always explore from the vertex of highest degree, with the edges chosen in descending order of degree of their end vertices. MBFSUE merges MBFS and the concept of unexplored edges, which says that as vertices are explored from, their degrees are thought of as getting depleted; therefore every vertex in the partial tree has a degree with respect to "unexplored" edges that is smaller than its degree in the original graph. Therefore, the new vertex considered is always the vertex of highest degree in G, the degree measured with respect to unexplored edges.

For purposes of this study only this three algorithms for generating fundamental cycles are investigated. The reader interested in a more comprehensive discussion of the subject should refer to Deo [1], Egyhazy [2], Paton [3], and Welch [4].

3.0 GRAPH REPRESENTATIONS

The choice of a particular graph representation is an impotant design decision in implementing algorithms to perform functions on graphs. We know intuitively that the performance characteristics of software, given the same set of graphs and algorithms, will be different for each graph representation implemented. It is this claim that we are set forth to prove in

the context defined by algorithms to generate fundamental cycles in graphs. The next three sections will define the terminology and notation for the three most common graph representations: Adjacency matrix, adjacency list and adjacency multilist.

3.1. Adjacency Matrix Graph Representation

In an adjacency matrix representation of a graph G the matrix entry (Vi, Vj) equals one if and only if there is an edge between vertex v_i and vertex v_j. Since for this investigation only simple graphs are considered, the main diagonals in the adjacency matrix will always equal zero. Accordingly, in the adjacency matrix representation of a graph, only the upper triangle of the adjacency matrix will be used to represent the graph.

3.2. Adjacency List Graph Representation

In the adjacency list representation of a graph G there will be one list for each vertex v_i in G. In each list the nodes represent the specific verticies adjacent to vertex v_j. Each node consists of two fields, INFO and LINK. Th INFO field contains the identifier of the next vertex that is also adjacent to vertex v_i.

3.3. Adjacency Multilist Graph Representation

In the adjacency multilist representation of a graph G there will be one list for each vertex in G. Each list has one node for each edge in G. The node represents an edge that has verticies v_i and v_j incident on a specific edge. Thus each node will be a member of two lists; one a member for vertex v_i and one a member for vertex v_j.

Each list node has five fields. Let M, INFOVI, INFOVJ, LINKVI, and LINKVJ denote these five fields. The first field M is an indicator of whether the edge has been inspected. The second field INFOVI contains the identifier of vertex v_i that is incident to the edge represented by the node. The third field INFOVJ contains the identifier of vertex v_j for which both v_i and v_j are incident to the edge represented by the node. The fourth field LINKVI contains a pointer to another node for which an edge is incident to vertex v_i. The fifth field LINKVJ contains a pointer to another node for which an edge is incident to vertex v_j.

3.5. Implementation of Graph Representations

All three graph representations described above were implemented in both IBM and VAX computers using the programming lan-

guage PASCAL. The choice of PASCAL eliminated any consideration to alternative implementations within a particular graph representation. Therefore, the emphasis was in finding the proportion of CPU time consumed in representing graph G as opposed to the execution of a particular algorithm to generate the set of fundamental cycles in G. Tests were conducted for twelve preselected graphs, using all three graph representations and the BCG, SDS, and MBFSUE algorithms. Details of the tests are found in Section 5.0.

4.0. GRAPHS USED IN TEST RUNS

Two sets of graphs were produced for the test runs, randomly generated and preselected from typical computer network configurations [5, 6, 7, 8]. These graphs are examples of the following network structures:

a) Distributed Network: This graph has 11 vertices and is configured so that no vertex is further away than three edges from any other vertex (Figure A).

b) Point to Point Long Haul Network: This graph has 41 vertices, with 8 vertices forming a ring structure. One of the ring vertices is connected to a subnet structure while the others are the roots of tree structures (Figure B).

c) Double Ring Network: This graph has a double ring like structure consisting of 12 vertices. Each ring vertex is connected to one and only one vertex of the other ring (Figure C).

d) Intersecting Loop: This graph has two distinct cycles with one vertex in common (Figure D).

f) Two Level Hierarchy with Five Regions: This graph, comprised of 17 vertices has two levels of hierarchy and five distinct regional topology (Figure F).

g) Interconnection Network of Networks: This graph has a complete subgraph of five vertices as the interconnecting network and five separate network structures connected to the complete subgraph by only one edge (Figure G).

h) Regular Network Representation: This graph has 20 verties with every vetex of degree for (Figure H).

i) Irregular Network Representation: This graph has 12 vertices with one star tree configuration as one subgraph (Figure I).

j) Linked Network Representation: This graph has 24 vertices with a "unibus" structure to which five subgraphs are connected (Figure J).

k) Distributed Network Representation: This graph has 128 vertices with a complete subgraph and four star like configurations. This network is similar in structure to the DATAPAC Packet Switched Network [5]. The graph is given in Figure K.

l) Subnet Representations: Two subnet graphs of 15 and 8 vertices are illustrated in Figure L and E respectively.

All of the above graphs are given pictorially in Appendix A. Figure 1.0 shows the number of nodes and edges for each graph, as well as the values for the nullity or number of fundamental cycles.

Graphs	No. Nodes	No. Edges	Nullity
A	10	13	4
B	41	49	9
C	12	18	7
D	15	16	2
E	8	9	2
F	17	21	5
G	38	57	20
H	16	32	17
I	12	12	1
J	21	25	5
K	18	20	3
L	15	19	5

Figure 1.0

5.0 IMPLEMENTATION AND TEST RESULTS

The BCG, SDS and MBFSUE algorithms were implemented using the adjacency matrix, one way adjacency list, and the adjacency multilist graph representations. Thus, there were a total of ten algorithm-graph representation combinations. Two different computers, an IBM 4341 VM/CMS and a VAX 11-780 VMS, were used in order to observe the performance of the algorithm graph representations running on two different machines. The computer language PASCAL was used to implement the algorithm-graph representation combinations. The code listings for the three algorithms and three graph representations are given in Appendix B.

For each of these combinations, three different CPU minute consumption measures occurred. The first measure was obtained by determining the CPU minutes consumed in the execution of a specific PASCAL program, hereby referred to as the "total" program, which implemented a particular algorithm-graph representation combination. The second CPU minute measure indicated the CPU minutes consumed in program initialization for ascertaining the fundamental cycles and the initialization of a specific graph representation data structure. This second measure was obtained by determining the CPU minutes consumed in the execution of a particular PASCAL program, hereby referred to as the "initialization" program, which implemented the initialization process. The

third CPU measure indicated the minutes consumed in traversing the graph and the determination of the average length of the fundamental cycle set. This third measure was obtained by simply subtractig the CPU minutes consumed by the "initialization" proram from the CPU minutes consumed by the "total" program.

The virtual CPU minutes consumed was used for measuring CPU minute consumption on the IBM 4341 VM/CMS. This quantity was obtained by involking a QUERY TIME command just prior to execution, immediately after execution and then subtractig the first from the second. On the other hand, the total CPU minutes consumed was used for measuring CPU minute consumption on the VAX-11/780 VMS. This quantity was obtained by involving a SHOW STATUS command just prior to executing a program, immediately after execution and then substracting the first from the second. Since the total CPU time on the VAX is influenced by the load on the system at the time of execution, the measures of CPU minute consumption were averaged over three executions, performed at different times of the day, of the same program.

The average length of the fundamental cycle was determined using the BCG, SDS and MBFSUE algorithms for all twelve graphs. Table I displays the individual results, with totals for each of the three algorithms. The SDS and MBFSUE algorithms yielded the minimal total fundamental cycle set averages of 57.6548 and 57.8402 respectively, while the BCG algorithm yielded 59.1548. This result, although expected should be considered as an indicator only, since there is some arbitrariness to be accounted for.

For example, the BCG algorithm selects an arbitrary starting vertex and subsequent vertices are chosen arbitrarely from these verticies that are adjacent to the last vertex explored from. The SDS algorithm selects the starting vertex of the highest degree and when there are several starting verticies of the highest degree in arbitrary choice is made. While the MBFSUE algorithm selects the starting vertex and all subsequent adjacent verticies according to the vertex with the highest degree. Therefore, with all three algorithms the arbitrary choices made in the initial labeling of the graph and those inherent to them lead to different average fundamental cycle lengths for the same graph G. It is conceivable therefore, that given a different initial labeling, the results for the same graph G turn out to be quite different. The extent of the variations is in itself a subject for further research.

A close examination of Table I shows that the MBFSUE algorithm gives a longer average length for fundamental cycles for graphs B, F and L. We conjecture that the reason for this lies partly in the fact that in all three of these graphs we find vertices of high degree forming long cycles. MBFSUE will find them before any of the shorter cycles are generated. This is not the case with the BCG or SDS algorithms, since both examine all adjacent vertices first regardless of their degrees, generating shorter cycles (if any in the vicinity) before considering the longer ones.

The SDS algorithm generated shorter, or at least the same, average fundamental cycle length for each of the twelve graphs. Thus, corraborating Deo's [1] claim about the superior performance of SDS over BCG. On the average, the SDS and MBFSUE algorithms performed similarly; for graphs A, C, G, H and J the MBFSUE did better while for graphs B, and L it was the other way around.

Table II presents for each graph the IBM 4341 VM/CMS virtual CPU minute consumption values for the BCG, SDS, and MBFS, algorithms as implemented using an adjacency matrix graph representation. Tables III and IV present the IBM 4341 VM/CMS virtual CPU minute consumption values for the BCG, SDS, and MBFS algorithms as implemented respectively with the one way adjacency list and adjacency multilist graph representations. Table V presents the same results as Tables II, III, and IV except the Table V shows that for each algorithm (BCG, SDS, and MBFSUE) the adjacency multilist graph representation implementation yielded the minimal expenditure of virtual CPU minutes, the one way adjacency list graph representation yielded the next lowest expenditure of virtual CPU minutes while the adjacency matrix graph representation yielded the highest expenditure of virtual CPU minutes. The difference in CPU minute consumption between the adjacency multilist representations and the one way adjacency list representations is due primarily to lower CPU minutes consumed in the traversal of the adjacency multilists vs. the traversal of the one way adjacency lists. The difference in CPU minute consumption between

the one way adjacency list representations and the adjacency matrix representations is due primarily to lower CPU minutes consumed in the program and graph representation initialization for the one way adjacency lists as compared to the program and graph representation initialization for the adjacency matricies.

Table VI presents for each graph the VAX-11 VMS total CPU minute consumption values for the BCG, SDS, and MBFSUE, algorithms as implemented using an adjacency matrix graph representation. Tables VII and VIII present the VAX-11 VMS total CPU minute consumption values for the BCG, SDS, and MBFSUE algorithms as implemented respectively with the one way adjacency list and adjacency multilist graph representations. Table IX presents the same results as Tables VI, VII, and VIII except the CPU minute measures are summed across all twelve graphs. Table IX shows that for each algorithm (BCG, SDS, and MBFSUE) the adjacency multilist graph representation implementation and the one way adjacency list graph representation implementation yielded the lowest expenditure of total CPU minutes while the adjacency matrix graph representation yielded the highest expenditure of VAX-11 total CPU seconds. The difference in VAX-11 total CPU minute consumption between the adjacency matrix graph representation and the other two graph representations is due primarily to higher CPU minutes consumed in the program and graph representation initialization for the adjacency matricies as compared to the program and graph representation initialization for the other two graph representation.

TABLE 1

AVERAGE LENGTH OF THE FUNDAMENTAL CYCLE SET AS FOUND BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE)

GRAPHS: / ALGORITHM	A	B	C	D	F	G	H	I	J	K	L	M	TOTAL
BCG	6.0000	4.1111	5.1429	8.0000	5.0000	4.0000	5.5294	3.0000	3.5714	3.0000	5.8000	6.0000	59.1548
SDS	5.0000	4.1111	5.1429	8.0000	5.0000	4.0000	5.5294	3.0000	3.5714	3.0000	5.8000	5.5000	57.6548
MBFSUE	4.7500	4.6667	4.8571	8.0000	6.2000	3.7000	4.8235	3.0000	3.1429	3.0000	6.2000	5.5000	57.8402

TABLE II

IBM 4341 VM/CMS VIRTUAL CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES
BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE) AND FOR THE ADJACENCY MATRIX GRAPH REPRESENTATION

GRAPHS:	A	B	C	D	F	G	H	I	J	K	L	M
ALGORITHM: BCG												
INITIALIZATION:	.05	.23	.05	.07	.07	.20	.07	.05	.10	.08	.07	.04
TRAVERSAL:	.03	.09	.04	.04	.04	.11	.08	.03	.06	.04	.05	.03
TOTAL:	.08	.32	.09	.11	.11	.31	.15	.08	.16	.12	.12	.07
ALGORITHM: SDS												
INITIALIZATION:	.05	.23	.06	.06	.07	.20	.07	.05	.11	.07	.06	.04
TRAVERSAL:	.03	.09	.04	.04	.05	.13	.09	.04	.05	.05	.06	.04
TOTAL:	.08	.31	.10	.10	.12	.33	.16	.09	.16	.12	.12	.08
ALGORITHM: MBFSUE												
INITIALIZATION:	.05	.24	.05	.07	.07	.21	.07	.06	.11	.07	.07	.05
TRAVERSAL:	.05	.11	.08	.04	.07	.16	.12	.04	.08	.07	.06	.05
TOTAL:	.10	.35	.13	.11	.14	.37	.19	.10	.19	.14	.13	.10

TABLE III

IBM 4341 VM/CMS VIRTUAL CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES
BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE) AND FOR THE ONE WAY ADJACENCY LIST GRAPH REPRESENTATION

GRAPHS:	A	B	C	D	F	G	H	I	J	K	L	M
ALGORITHM: BCG												
INITIALIZATION:	.05	.11	.05	.05	.07	.11	.06	.05	.07	.06	.06	.05
TRAVERSAL:	.05	.07	.04	.05	.05	.11	.11	.05	.07	.04	.06	.04
TOTAL:	.10	.18	.09	.10	.12	.22	.17	.10	.14	.10	.12	.09
ALGORITHM: SDS												
INITIALIZATION:	.05	.11	.05	.05	.06	.11	.07	.05	.08	.06	.06	.05
TRAVERSAL:	.05	.08	.07	.06	.06	.11	.10	.04	.06	.05	.06	.05
TOTAL:	.10	.19	.12	.11	.12	.22	.17	.09	.14	.11	.12	.10
ALGORITHM: MBFSUE												
INITIALIZATION:	.05	.11	.05	.05	.06	.10	.07	.05	.07	.06	.06	.05
TRAVERSAL:	.06	.12	.09	.06	.08	.17	.13	.05	.09	.06	.08	.05
TOTAL:	.11	.23	.14	.11	.14	.27	.20	.10	.16	.12	.14	.10

TABLE IV

IBM 4341 VM/CMS VIRTUAL CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES
BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE) AND FOR THE ADJACENCY MULTILIST GRAPH REPRESENTATION

GRAPHS:	A	B	C	D	F	G	H	I	J	K	L	M
ALGORITHM: BCG												
INITIALIZATION:	.05	.08	.05	.05	.05	.09	.06	.05	.07	.05	.06	.04
TRAVERSAL:	.03	.06	.04	.03	.04	.09	.08	.02	.04	.03	.03	.03
TOTAL:	.08	.14	.09	.08	.09	.18	.14	.07	.11	.08	.09	.97
ALGORITHM: SDS												
INITIALIZATION:	.05	.08	.05	.05	.06	.10	.06	.04	.06	.06	.05	.04
TRAVERSAL:	.04	.07	.05	.03	.04	.09	.09	.04	.06	.03	.06	.04
TOTAL:	.09	.15	.10	.08	.10	.19	.15	.08	.12	.09.	.11	.08
ALGORITHM: MBFSUE												
INITIALIZATION:	.04	.09	.05	.05	.05	.09	.05	.05	.07	.05	.05	.04
TRAVERSAL:	.05	.09	.06	.04	.06	.13	.11	.03	.07	.05	.06	.05
TOTAL:	.09	.18	.11	.09	.11	.22	.16	.08	.14	.10	.11	.09

TABLE V

SUMMARY ACROSS TWELVE GRAPHS OF THE IBM 4341 VM/CMS VIRTUAL CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE) AND BY ALTERNATIVE GRAPH REPRESENTATIONS (ADJACENCY MATRIX, ONE WAY ADJACENCY LISTS, ADJACENCY MULTILISTS)

	BCG	SDS	MBFSUE
ADJACENCY MATRIX GRAPH REPRESENATION			
INITIALIZATION:	1.08	1.07	1.12
TRAVERSAL:	.64	.71	.93
TOTAL:	1.72	1.78	2.05
ONE WAY ADJANCENCY LIST GRAPH REPRESENTATION			
INITIALIZATION:	.79	.80	.78
TRAVERSAL:	.74	.79	1.04
TOTAL:	1.53	1.59	1.82
ADJACENCY MULTILIST GRAPH REPRESENTATION			
INITIALIZATION:	.70	.70	.68
TRAVERSAL:	.52	.64	.80
TOTAL:	1.22	1.34	1.48

TABLE VI

VAX-11/VMS CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES
BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE) AND FOR THE ADJACENCY MATRIX GRAPH REPRESENTATION

GRAPHS:	A	B	C	D	F	G	H	I	J	K	L	M
ALGORITHM: BCG												
INITIALIZATION:	.29	.66	.30	.32	.34	.59	.33	.30	.40	.34	.30	.29
TRAVERSAL:	.03	.09	.04	.04	.05	.05	.08	.01	.06	.04	.04	.03
TOTAL:	.32	.75	.34	.36	.39	.64	.41	.31	.46	.38	.34	.32
ALGORITHM: SDS												
INITIALIZATION:	.31	.66	.30	.31	.33	.62	.31	.29	.41	.37	.32	.25
TRAVERSAL:	.02	.13	.04	.06	.04	.12	.09	.04	.06	.05	.04	.08
TOTAL:	.33	.79	.34	.37	.37	.74	.40	.33	.47	.42	.36	.33
ALGORITHM: MBFSUE												
INITIALIZATION:	.27	.67	.29	.33	.34	.64	.31	.31	.41	.33	.31	.29
TRAVERSAL:	.06	.14	.08	.03	.07	.11	.12	.04	.09	.06	.07	.05
TOTAL:	.33	.81	.37	.36	.41	.75	.43	.35	.50	.39	.38	.34

TABLE VII

VAX-11/VMS CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES
BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE) AND FOR THE ONE WAY ADJACENCY LIST GRAPH REPRESENTATION

GRAPHS:	A	B	C	D	F	G	H	I	J	K	L	M
ALGORITHM: BCG												
INITIALIZATION:	.27	.33	.25	.29	.27	.31	.29	.26	.30	.27	.28	.27
TRAVERSAL:	.03	.06	.05	.01	.06	.09	.08	.03	.03	.04	.02	.03
TOTAL:	.30	.39	.30	.30	.33	.40	.37	.29	.33	.31	.30	.30
ALGORITHM: SDS												
INITIALIZATION:	.27	.31	.29	.27	.30	.33	.27	.26	.29	.28	.28	.27
TRAVERSAL:	.05	.06	.04	.06	.02	.04	.09	.01	.05	.06	.04	.05
TOTAL:	.32	.38	.33	.33	.32	.37	.36	.27	.34	.34	.32	.32
ALGORITHM: MBFSUE												
INITIALIZATION:	.27	.34	.29	.28	.30	.32	.31	.25	.30	.28	.28	.28
TRAVERSAL:	.07	.08	.04	.04	.06	.15	.08	.06	.06	.05	.06	.04
TOTAL:	.34	.42	.33	.32	.36	.47	.39	.31	.36	.33	.34	.32

TABLE VIII

VAX-11/VMS CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES
BY ALTERNATIVE ALGORITHMS (BCG, SDS, MBFSUE) AND FOR THE ADJACENCY MULTILIST GRAPH REPRESENTATION

GRAPHS:	A	B	C	D	F	G	H	I	J	K	L	M
ALGORITHM: BCG												
INITIALIZATION:	.25	.29	.21	.27	.25	.27	.28	.25	.26	.28	.26	.24
TRAVERSAL:	.05	.04	.10	.05	.07	.07	.08	.06	.09	.02	.07	.07
TOTAL:	.30	.33	.31	.32	.32	.34	.36	.31	.35	.30	.33	.31
ALGORITHM: SDS												
INITIALIZATION:	.25	.30	.27	.27	.29	.28	.29	.29	.27	.27	.29	.25
TRAVERSAL:	.07	.07	.05	.05	.02	.20	.07	.01	.05	.03	.04	.06
TOTAL:	.32	.37	.32	.32	.31	.38	.36	.30	.32	.30	.33	.31
ALGORITHM: MBFSUE												
INITIALIZATION:	.26	.31	.25	.28	.28	.29	.29	.25	.28	.28	.27	.28
TRAVERSAL:	.05	.08	.06	.06	.06	.16	.09	.07	.07	.03	.05	.03
TOTAL:	.31	.39	.31	.34	.34	.45	.38	.32	.35	.31	.32	.31

TABLE IX

SUMMARY ACROSS TWELVE GRAPHS OF THE VAX-11/VMS CPU MINUTES CONSUMED IN ASCERTAINING FUNDAMENTAL CYCLES BY ALTERNATIVE ALGORITHMS (BDG, SDS, MBFSUE) AND BY ALTERNATIVE GRAPH REPRESENTATIONS (ADJACENCY MATRIX, ONE WAY ADJACENCY LISTS, ADJACENCY MULTILISTS)

	BCG	SDS	MBFSUE
ADJACENCY MATRIX GRAPH REPRESENTATION			
INITIALIZATION:	4.46	4.48	4.50
TRAVERSAL:	.56	.77	.92
TOTAL:	5.02	5.25	5.42
ONE WAY ADJACENCY LIST GRAPH REPRESENTATION			
INITIALIZATION:	3.39	3.43	3.50
TRAVERSAL:	.53	.57	.79
TOTAL:	3.92	4.00	4.29
ADJACENCY MULTILIST GRAPH REPRESENTATION			
INITIALIZATION:	3.11	3.32	3.32
TRAVERSAL:	.77	.62	.81
TOTAL:	3.88	3.94	4.13

6.0 Discussion of Results

A substantial limitation of the results reported herein is that these results are applicable only to the twelve graphs selected for study and which appear in Appendix A. The graphs were selected on the basis of intuitive judgements of the presence of different graph properties (blocks, colorability, rings, high density, low density). No attempt was made to draw a random sample from a population of all possible graphs.

One of the main findings of this research effort was that the choice of graph representation in the implementation of a fundamental cycle generation algorithm substantially influences the CPU time consumed. Indeed comparisons of CPU time expenditures by the BCG, SDS, and MBFSUE algorithms must not be confounded by the type of graph representation used in the implementation of the algorithms. Across the BCG, SDS, and MBFSUE algorithms the adjacency multilist and the one way adjacency list graph representation implementations were found to require less CPU time than did the adjacency matrix graph representation implementations.

REFERENCES

1. Deo, N., Prabhu, G. M., and Krishnamoorthy, K. S. Algorithms for generating fundamental cycles in a graph. ACM Trans. Mathematical Software 8, 1 (Mar. 1982), 26-42.

2. Egyhazy, C. Efficient algorithms to determine a set of fundamental cycles of minimum average length in a graph. Technical Report CS-TR824. Computer Science Department, Virginia Polytechnic Institute and State University.

3. Paton, K. An algorithm for finding a fundamental set of cycles of a graph. Commu. ACM 12, 9 (Sept. 1969), 514-518.

4. Welch, J. T., Jr. A mechanical analysis of the cyclic structure of undirected linear graphs. J. ACM 13.2 (1966), 205-210.

5. Doll, R. D. Data Communications: Facilities, Networks and Systems Design, John Wiley & Sons, Inc. (1978).

6. Booth, T. L. Digital Networks and Computer Systems, John Wiley & Sons, Inc. (1978).

7. Martin, J. Teleprocessing Network Organization, Prentice-Hall, Inc. (1970).

8. Tanenbaum, A. S. Computer Networks, Prentice-Hall, Inc. (1981).

9. Harary, F. Graph Theory, Addison-Wesley (1972).

APPENDIX A

A SPECIAL CLASS OF GRAPHS

This appendix contains the topologies for the twelve graphs used in this research effort.

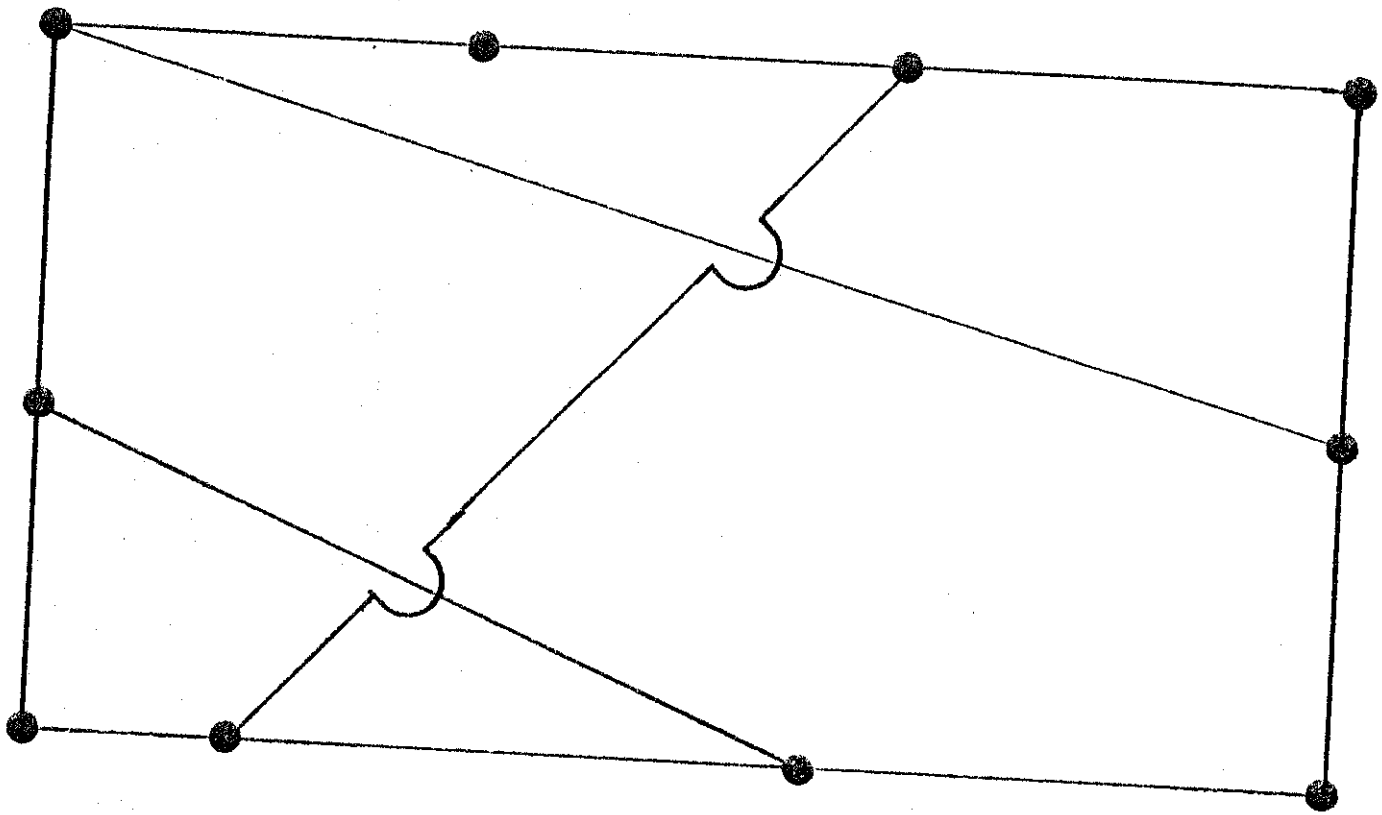

FIGURE A

Distributed Network

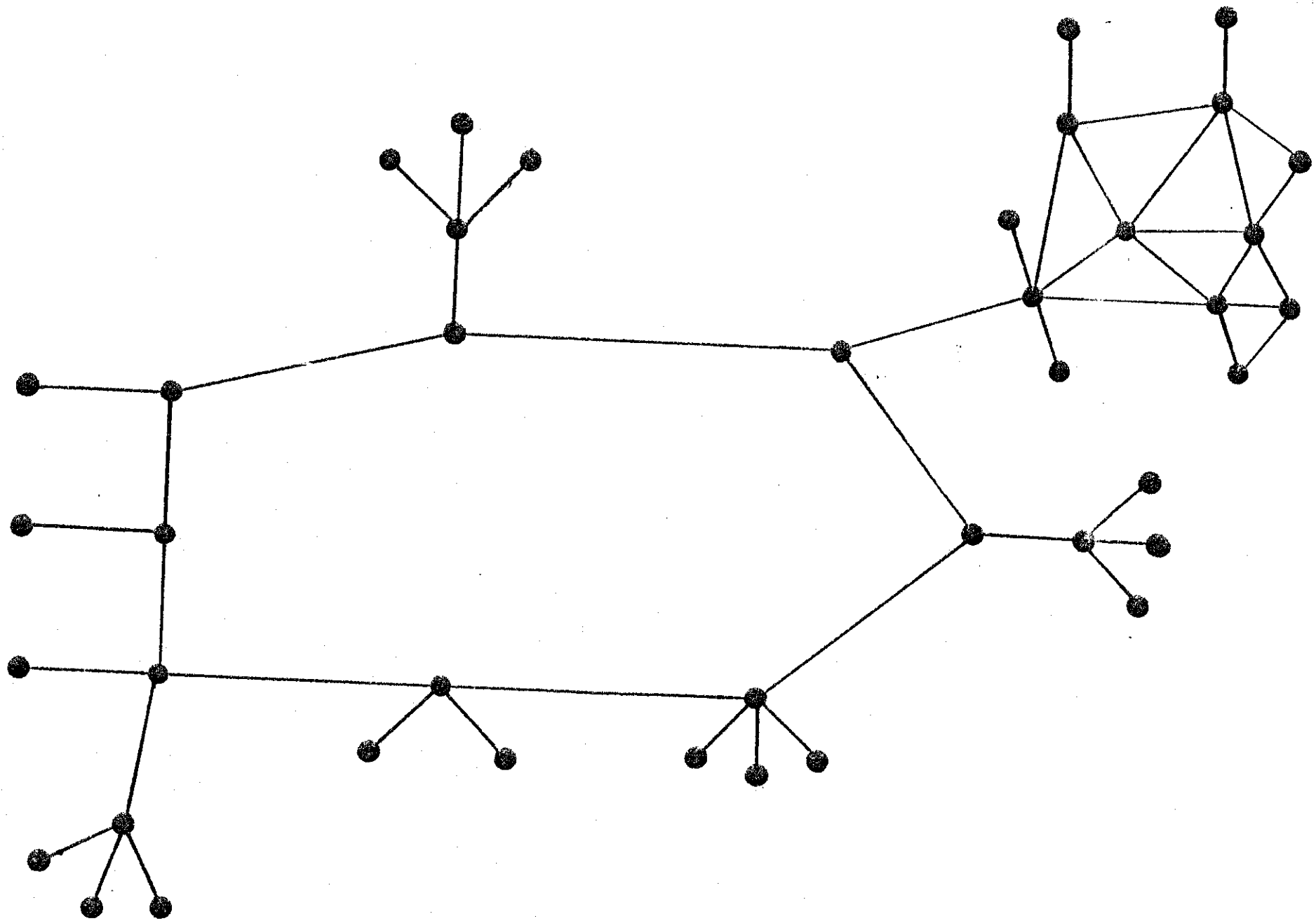

FIGURE B

Point to Point Long Haul Network

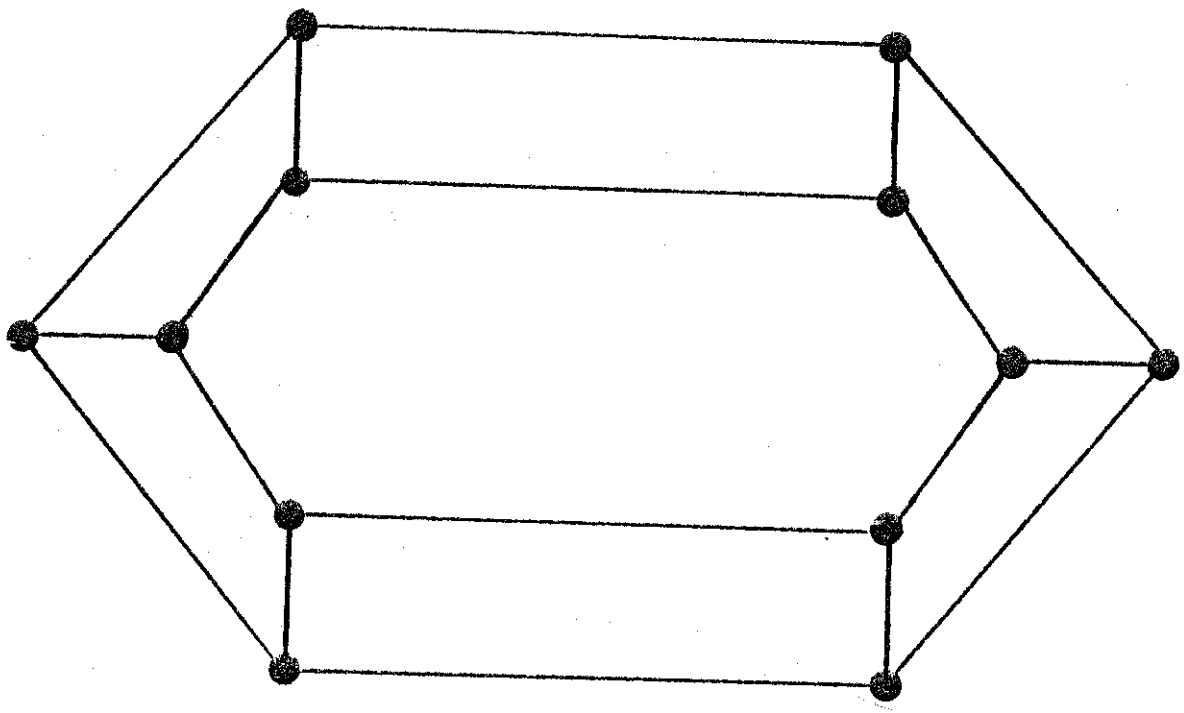

FIGURE C

Double Ring Like Network

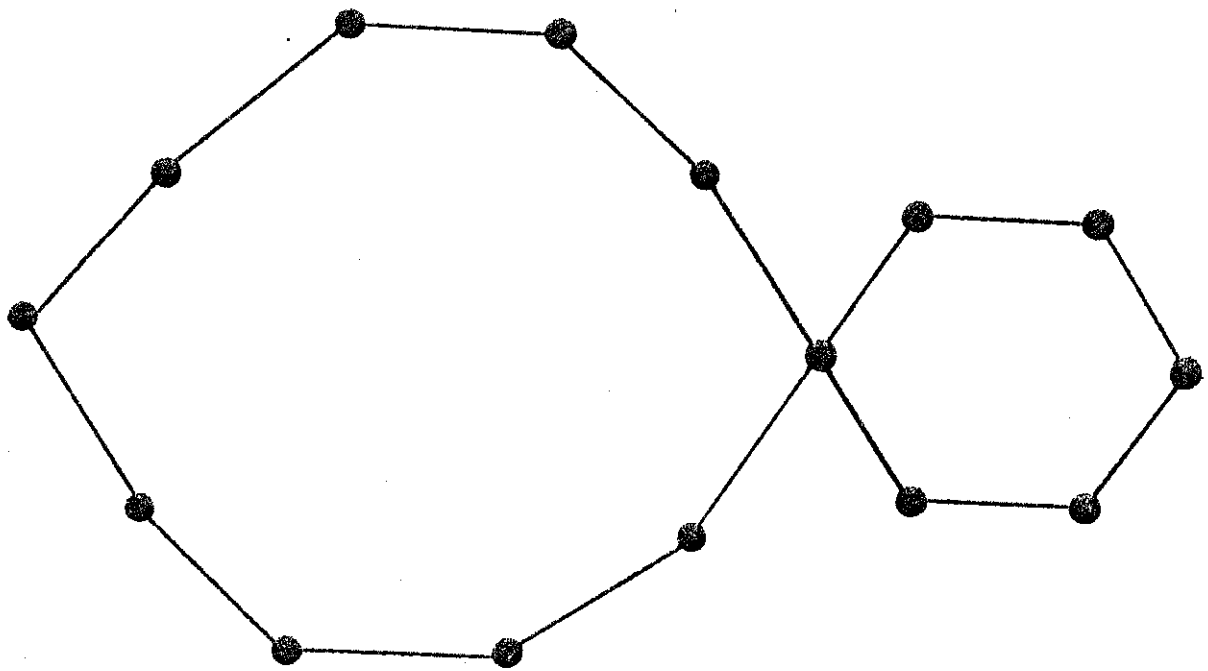

FIGURE D

Intersecting Loop

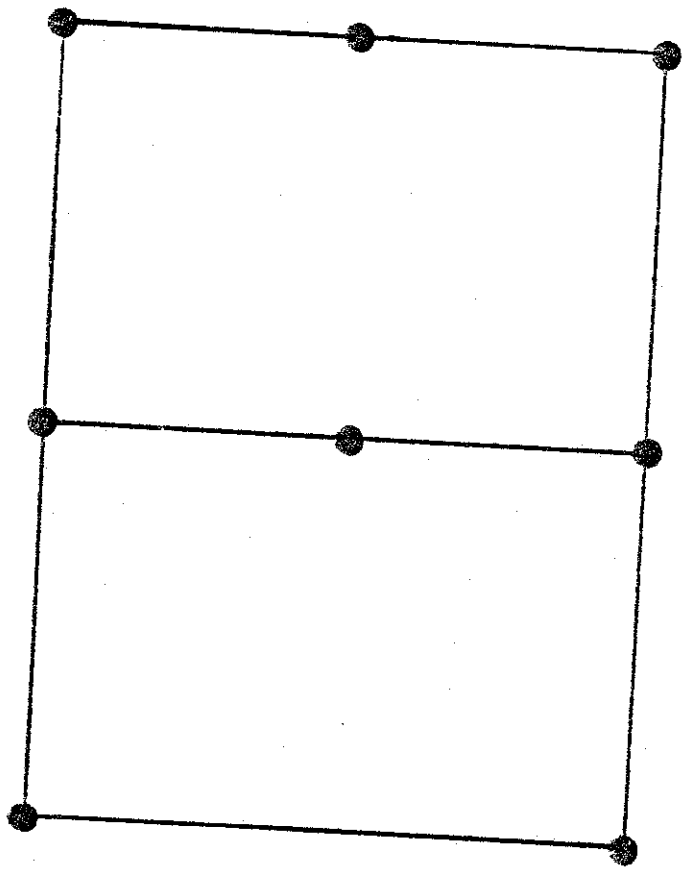

FIGURE E

Subnet 2

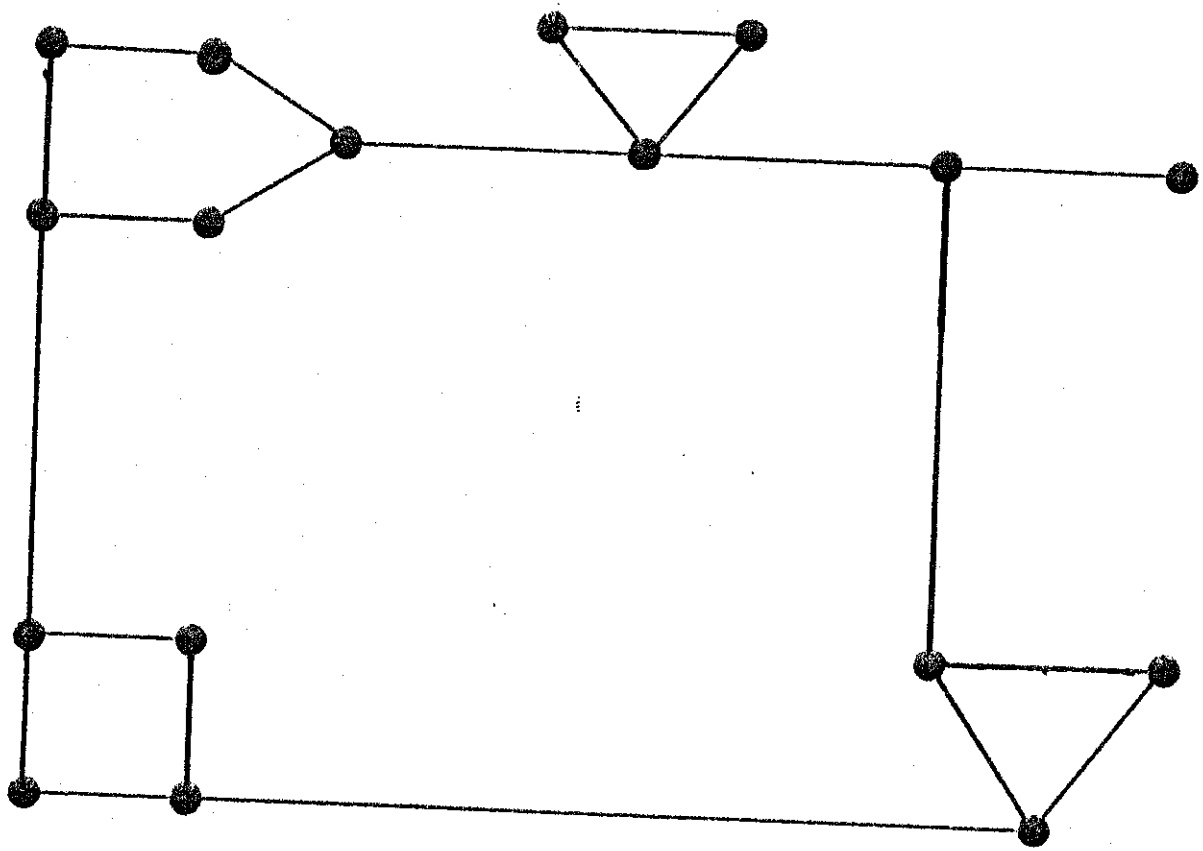

FIGURE F

Two Level Hierarchy With Five Regions

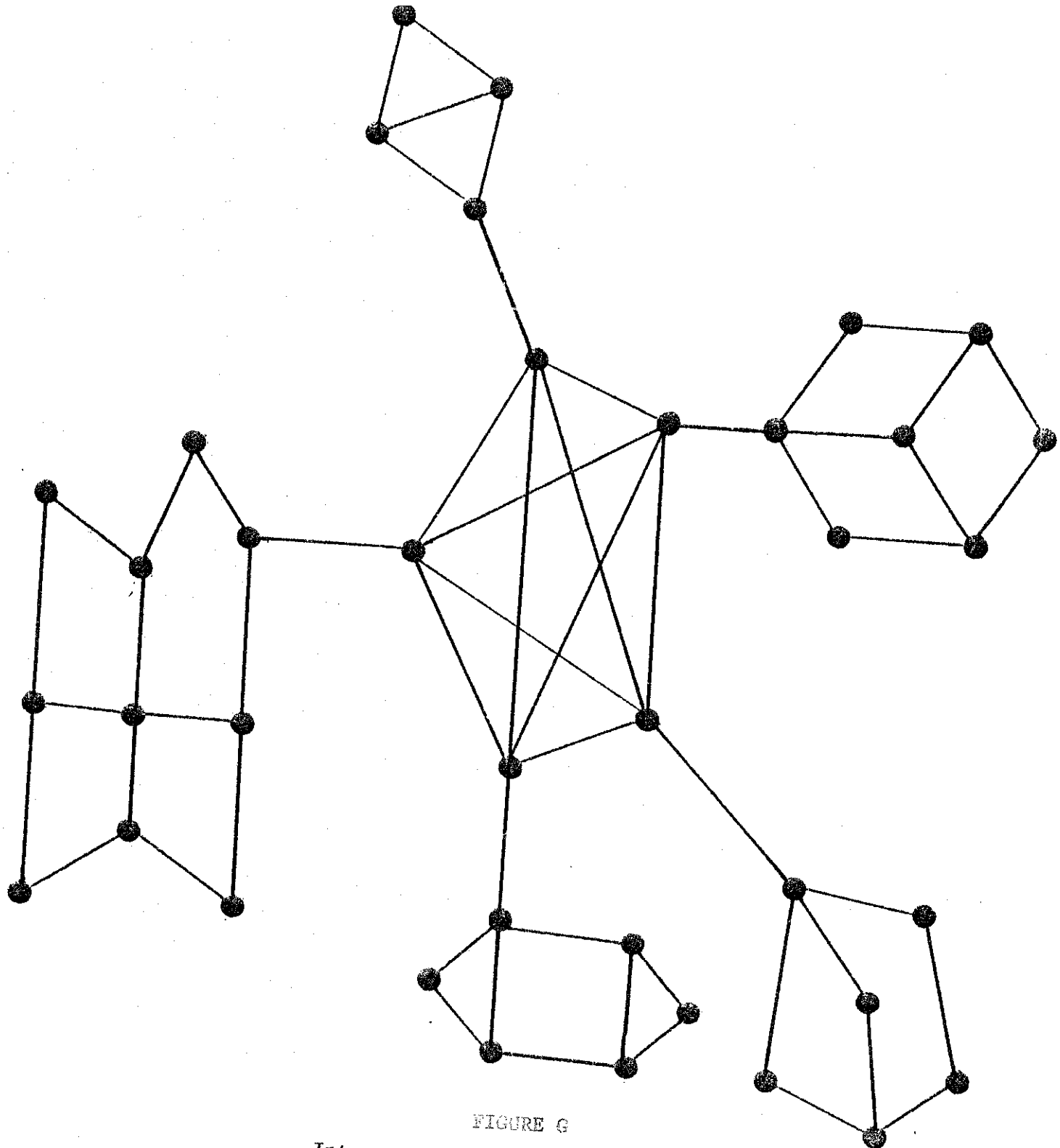

FIGURE G

Interconnection Network of Networks

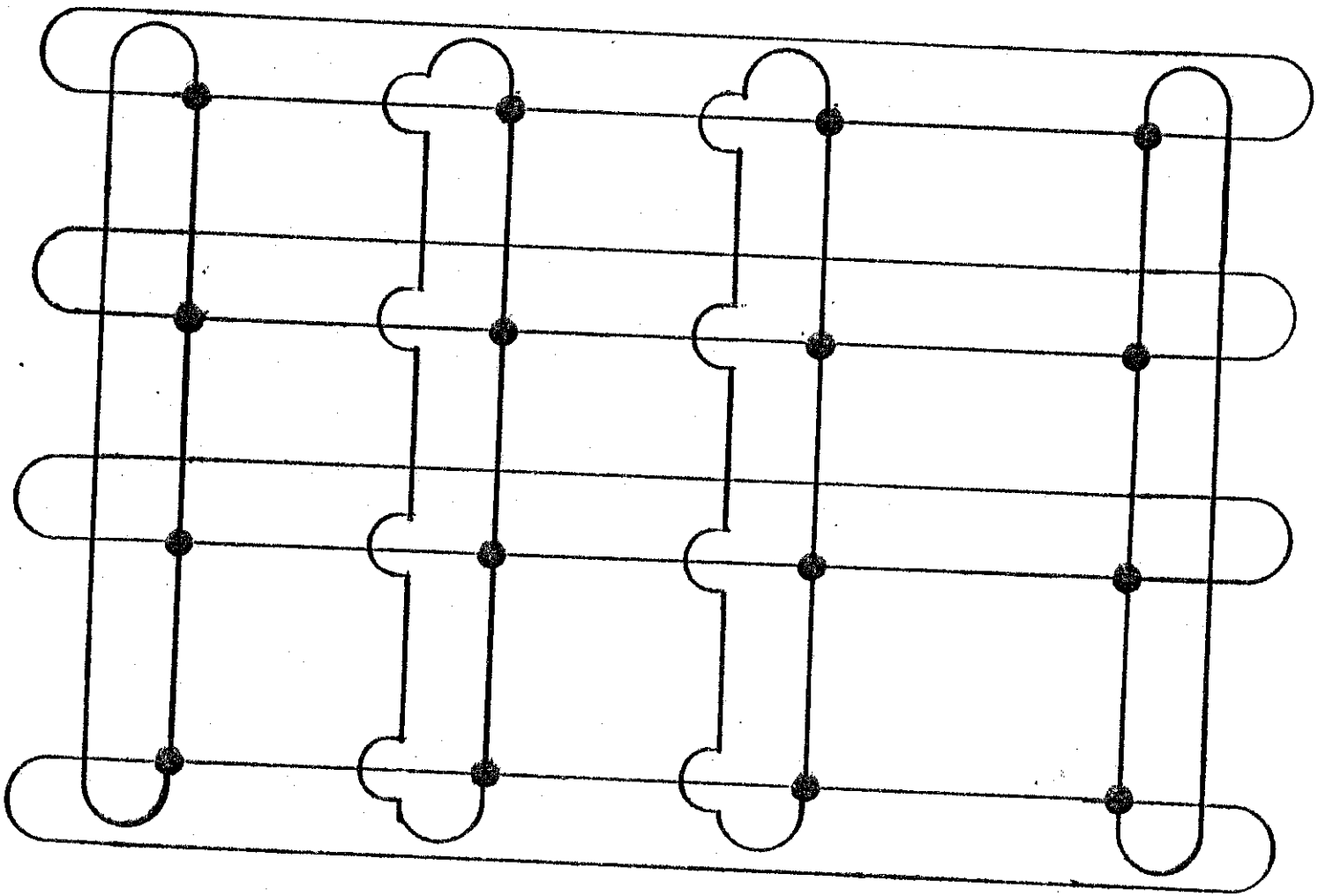

FIGURE H

Regular Network Representation

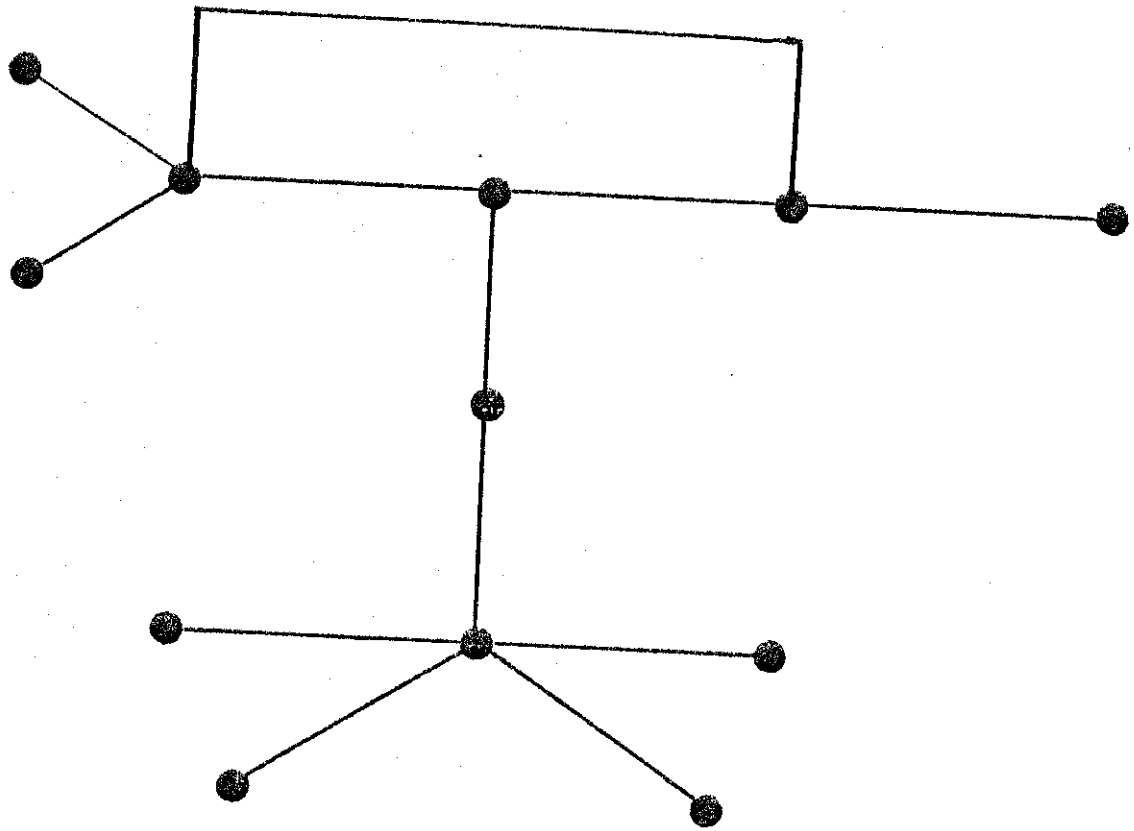

FIGURE I

Irregular Network Representation

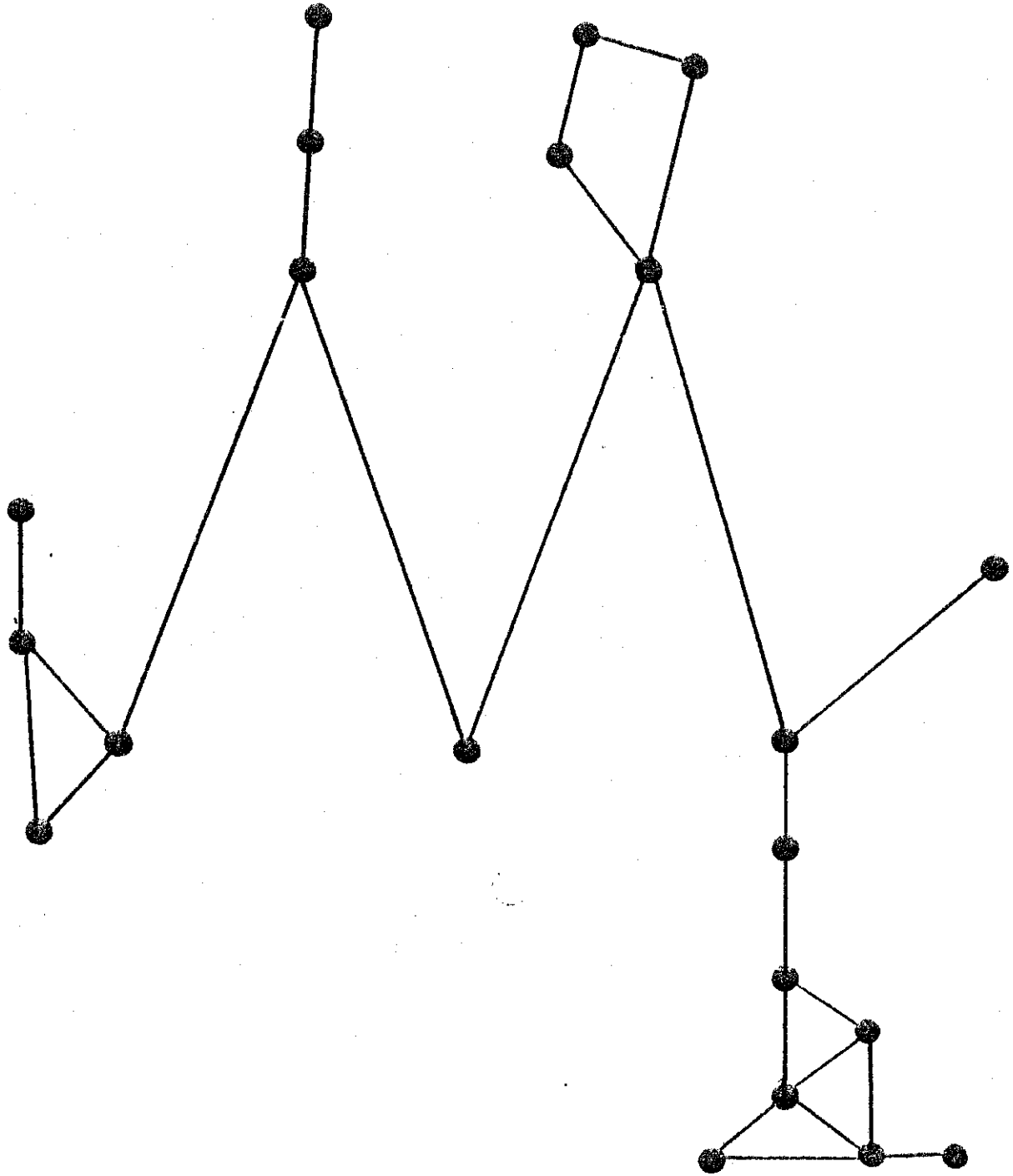

FIGURE J

Linked Network Representation

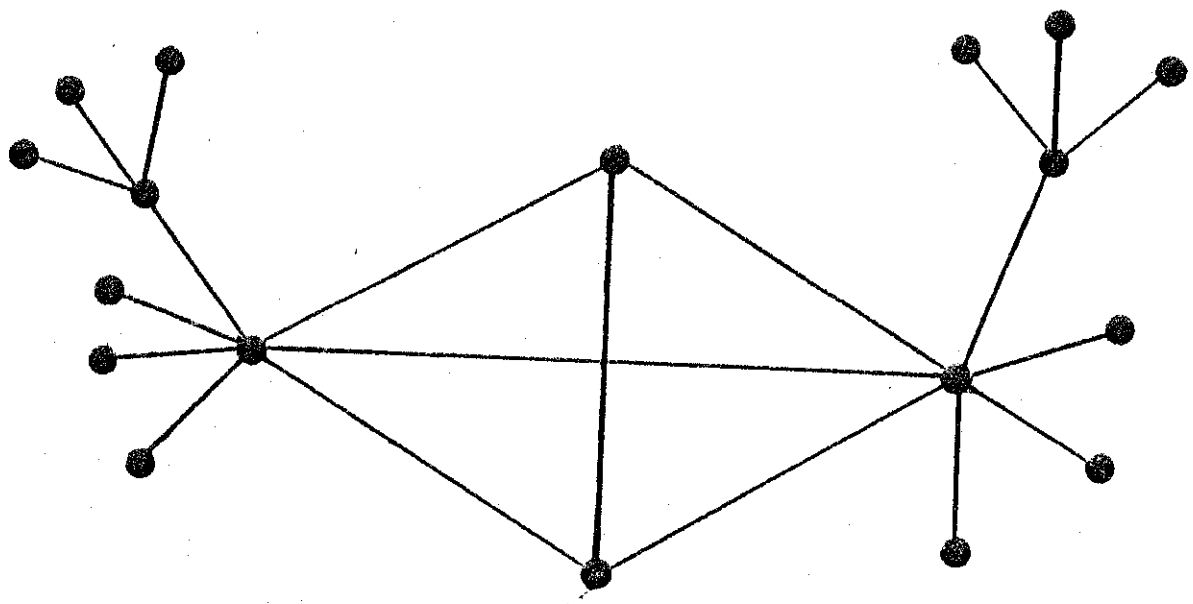

FIGURE K

DATAPAC Packet Switched Network

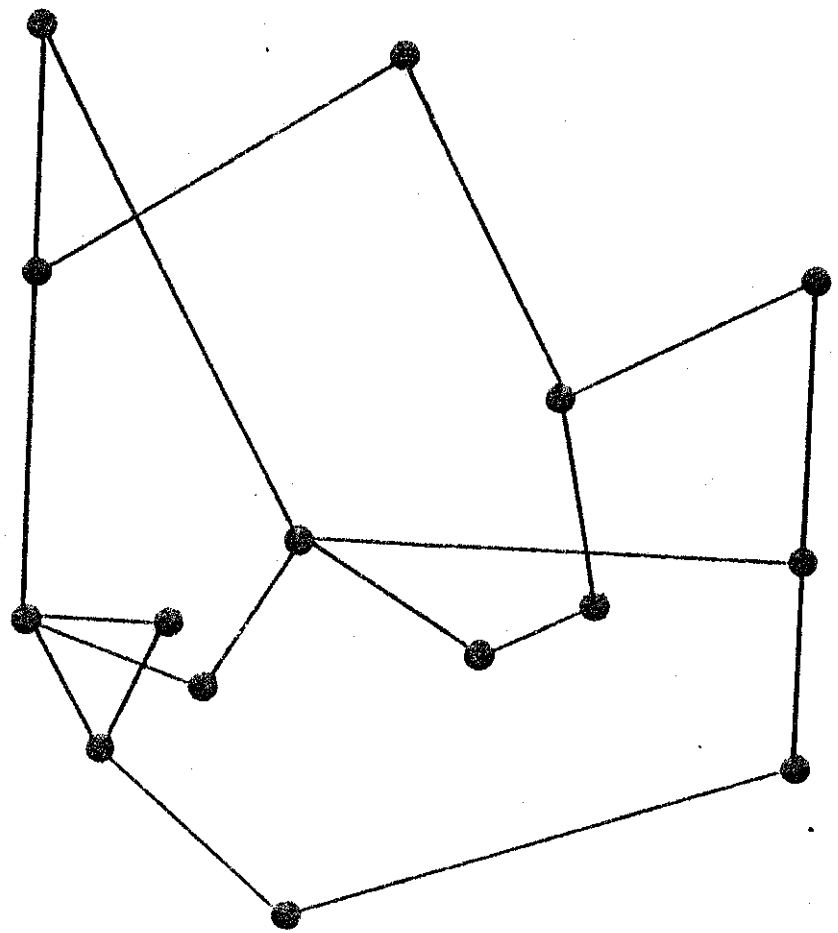

FIGURE L

Subnet 1