M.I.T. CIVIL ENGINEERING SYSTEMS LABORATORY

NEW LEVEL OF HIERARCHY

HIDECS 2: A COMPUTER PROGRAM FOR THE HIERARCHICAL DECOMPOSITION OF A SET WITH AN ASSOCIATED LINEAR GRAPH

Christopher Alexander
Marvin L. Manheim

M.I.T. CIVIL ENGINEERING SYSTEMS LABORATORY

RESEARCH REPORT R62-2

DEPARTMENT OF CIVIL ENGINEERING

CIVIL ENGINEERING SYSTEMS LABORATORY

HIDECS 2:
A COMPUTER PROGRAM FOR THE HIERARCHICAL DECOMPOSITION OF A SET WHICH HAS AN ASSOCIATED LINEAR GRAPH

by

Christopher Alexander,

Society of Fellows, Harvard University,

and

Marvin L. Manheim,

Department of Civil Engineering, M.I.T.

Publication No. 160
June, 1962

Sponsored by: Massachusetts Department of Public Works
In cooperation with: U. S. Bureau of Public Roads
Contract 1017 Mass. HPS 1(16)

School of Engineering
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

Acknowledgements

The research of which this report is a part was supported by the Bureau of Public Roads of the U.S. Department of Commerce and by the Massachusetts Department of Public Works. The computer and associated facilities were made available by the M.I.T. Computation Center.

The work was performed at the Civil Engineering Systems Laboratory at M.I.T. The authors gratefully acknowledge the suggestions and criticisms freely offered by Professors A. Scheffer Lang, Charles L. Miller, and Paul O. Roberts; however, all statements made herein are the responsibility of the authors.

The diagrams were prepared by Miss Candace Allen, of the Civil Engineering Systems Laboratory.

CONTENTS

FIGURES

FIGURES (Cont)

I. INTRODUCTION

A. ABSTRACT AND INTRODUCTION

The program discussed in this report was developed at the Civil Engineering Systems Laboratory, M.I.T. for application to the analysis of several problems in highway engineering. The nature of the analytical methods and of the specifications of the program allow for broad application to other subjects.

The program is used to analyze the structure of a linear graph, or topological one-complex, as it is also termed. Such a graph consists of just two types of elements: vertices, and non-directed links connecting specified pairs of vertices. The input to this program is the matrix description of the graph. According to a criterion derived from assumptions about the graph structure and its information-theoretic properties, the set of vertices can be divided into two or more subsidiary sets of vertices, called subsets, so that each subset still has an associated graph. The program performs such a partition on the input set of vertices, and then in turn partitions the resulting sets. This process is repeated, successivly decomposing the original vertex set into smaller and smaller subsets, until it has been completely decomposed into its constituent vertices.

The set of sets which results from the successive partitions is ordered naturally as a "tree." This tree specifies the order in which the subsets can be recombined to produce the original graph. In the application for which this program

was developed, the vertices of the graph represent the requirements of a design problem. The tree defined by the program's output specifies an order in which the designer should consider the requirements he tries to meet in the process of evolving a design.

This report describes the program, the algorithms upon which it is based, operational procedures, and certain possible modifications.

B. APPLICATION OF THE PROGRAM

This program was developed for use in the analysis of certain types of design problems, according to the theory put forward by Christopher Alexander, in "NOTES ON THE SYNTHESIS OF FORM" (Ph.D. thesis, Harvard University, 1962). The theory has been applied to two highway engineering problems: the design of a highway interchange, and the selection of a location for a highway.*

All design problems contain two kinds of element:

(1) Requirements

(2) Interactions between requirements.

The requirements which the design has to meet are represented as vertices of a graph. The purpose of the design process is to select a design (in the case of our examples, for a highway or an interchange) which fulfills these requirements. The difficulty of achieving a design which satisfies such a list of requirements is due to the fact that requirements conflict with one another: some requirements place demands upon the design which are contradictory to the demands of other requirements. The presence of these interactions between pairs of requirements in a specific design program is represented by links between the vertices corresponding to the particular requirements. The set of vertices, and the set of links together define a graph;

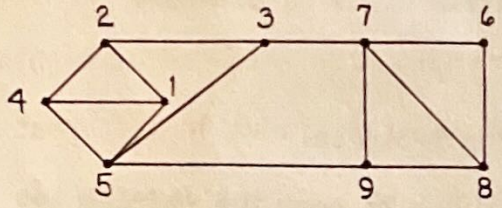
---

*See the reports on these projects by Alexander and Manheim, also issued by the Civil Engineering Systems Laboratory, M.I.T.: THE DESIGN OF HIGHWAY INTERCHANGES: AN EXAMPLE OF A GENERAL METHOD FOR ANALYSING ENGINEERING DESIGN PROBLEMS, and THE USE OF DIAGRAMS IN HIGHWAY ROUTE LOCATION: AN EXPERIMENT.

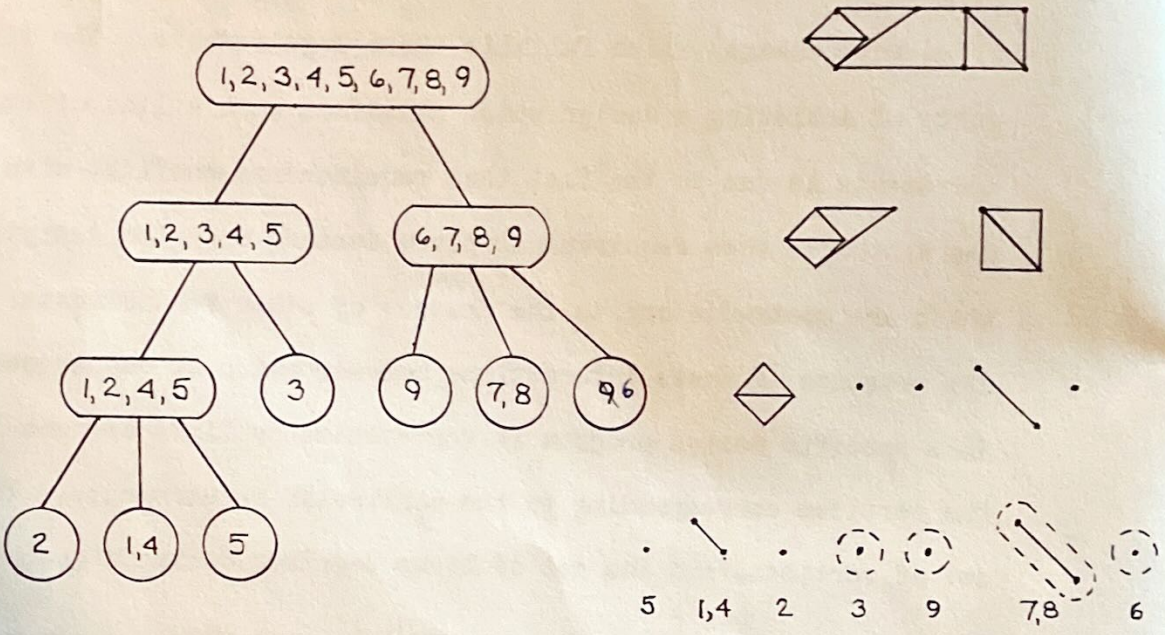
Figure 1

EXAMPLE OF A GRAPH AND ITS TREE

a) A GRAPH



b) ITS TREE



that graph, by virtue of the correspondence between vertices and requirements, and the correspondence between links and interactions of requirements,represents, for the purpose of this analysis, the structure of the design problem.

The input to the program is a graph; the output is a tree, a hierarchical ordering of the graph's vertex set and its partitioned subsets. Because of the correspondence between the graph and the problem, the tree which is obtained by the program provides an orderly scheme for dealing with the requirements posed by a particular problem. The tree specifies which requirements are to be considered together and the order in which different groups of requirements are to be combined and considered. See Figure 1.

C. MACHINE SPECIFICATION

This program has been debugged and run on an IBM 709 at the M.I.T. Computation Center. This machine uses 36-bit words, has a memory capacity of 32,767 words, and has three index registers. The program is designed to be executed under the control of the Fortran Monitor System in use at the Center during the second half of 1961. For information as to peculiarities of the M.I.T. installation which might prove critical in running this program on another machine, see further the M.I.T. Computation Center Procedures Handbook, 1961.

The program has also been used for production runs on an IBM 7090 at the Smithsonian Astrophysical Observatory, Cambridge, under the control of an M.I.T. system tape. No changes in the program were required.

It is not possible to give a rule for estimating the running time required for any specific analysis.

D. REFERENCES

Alexander, Christopher, NOTES ON THE SYNTHESIS OF FORM. Unpublished Ph.D. thesis. Harvard University (1962).

Alexander, Christopher and Marvin L. Manheim, THE DESIGN OF HIGHWAY INTERCHANGES: AN EXAMPLE OF A GENERAL METHOD FOR ANALYSING ENGINEERING DESIGN PROBLEMS. Cambridge, Mass.: Civil Engineering Systems Laboratory, M.I.T. (1962).

_____, THE USE OF DIAGRAMS IN HIGHWAY ROUTE LOCATION: AN EXPERIMENT. Cambridge, Mass.: Civil Engineering Systems Laboratory, M.I.T. (1962).

II. DESCRIPTION OF THE PROGRAM

This description is divided into three major sections. The first describes the operational structure of the program, as comprising a package of subprograms with specific functions. This serves as an introduction to the dictionary of subprograms in Appendix B. The second section of the description discusses the machine representation of a graph. The third section of the description discusses the body of the program, the algorithms actually used in the analysis of graphs. These algorithms are discussed in sufficient detail to introduce the actual program listing, included as Appendix G of the report.

A. GENERAL DESCRIPTION

The program consists of three groups of subprograms. The first, or Preliminary, group prepares for the execution of other subprograms by reading in several parameters, generating other standard parameters, and setting variables which control the execution of particular loops in other subprograms. This group has five subprograms: INPAR, GENER, SET8, SET9, and SET11.

The second, or Data, group of subprograms is concerned with reading in the binary data matrix (representing the graph to be analyzed), and putting it in appropriate form for the analysis. Several different operations are performed by this set of programs: INDAT reads in the data as it is punched on cards, CNDAT converts the inputted data from its input form into the format in which the other subprograms can operate upon it, and SYMET checks the data for inconsistencies which may arise in the pre-computer preparation of the data. With the results of the Preliminary and Data groups of subprograms, the analysis proper can be begun.

The group of subprograms which actually performs the analysis of the graph consists of seven programs actually involved in the analysis, and six which print out the results and comments. LCTRL is the major control program for the Analysis group; it controls the course of the partitioning iterations, the manner of storage of the results, and the selection of one of two sequences of subprograms.

Figure 2

GROUPING OF SUBPROGRAMS

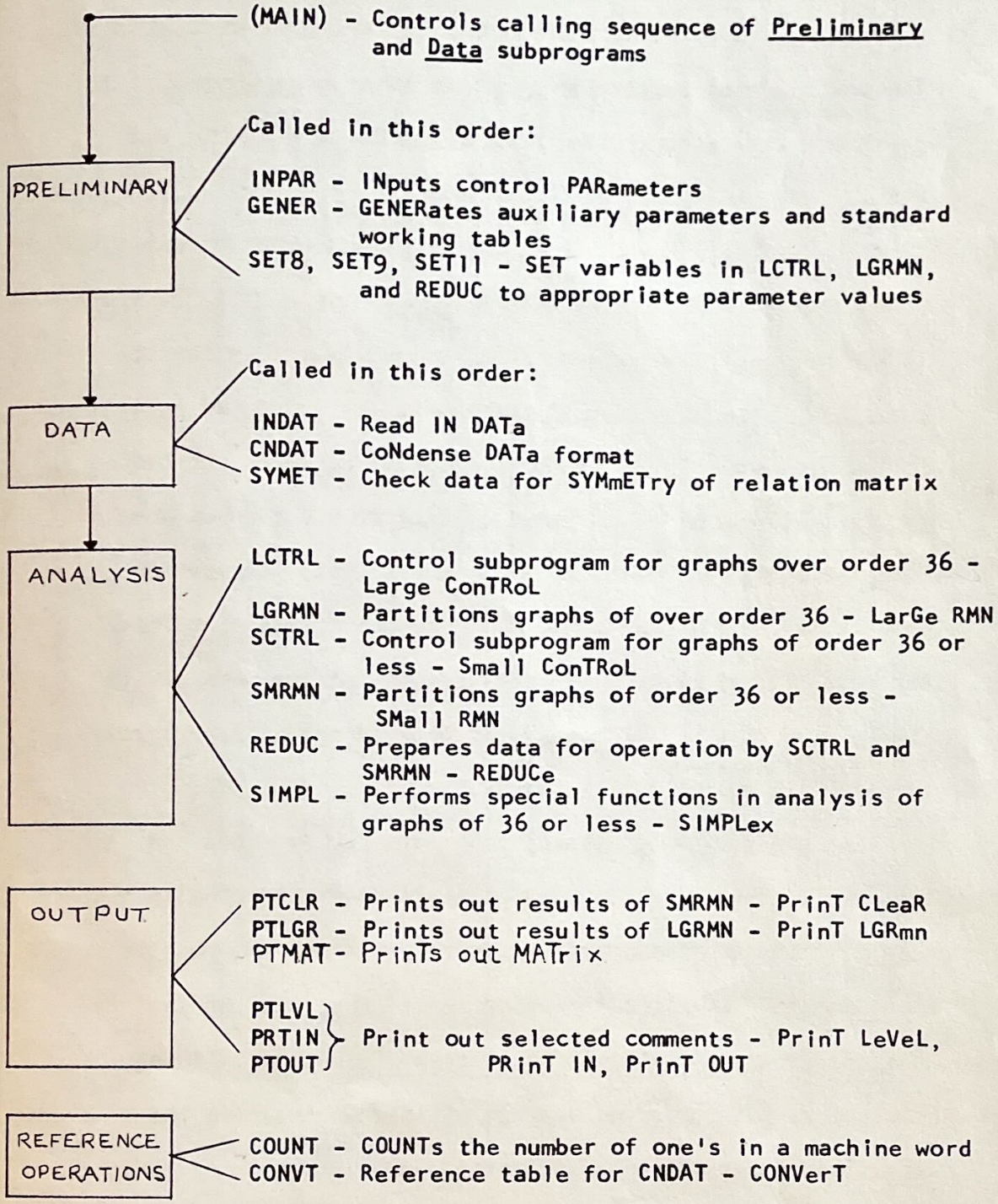


FIGURE 3

SEQUENCE OF DATA OPERATIONS

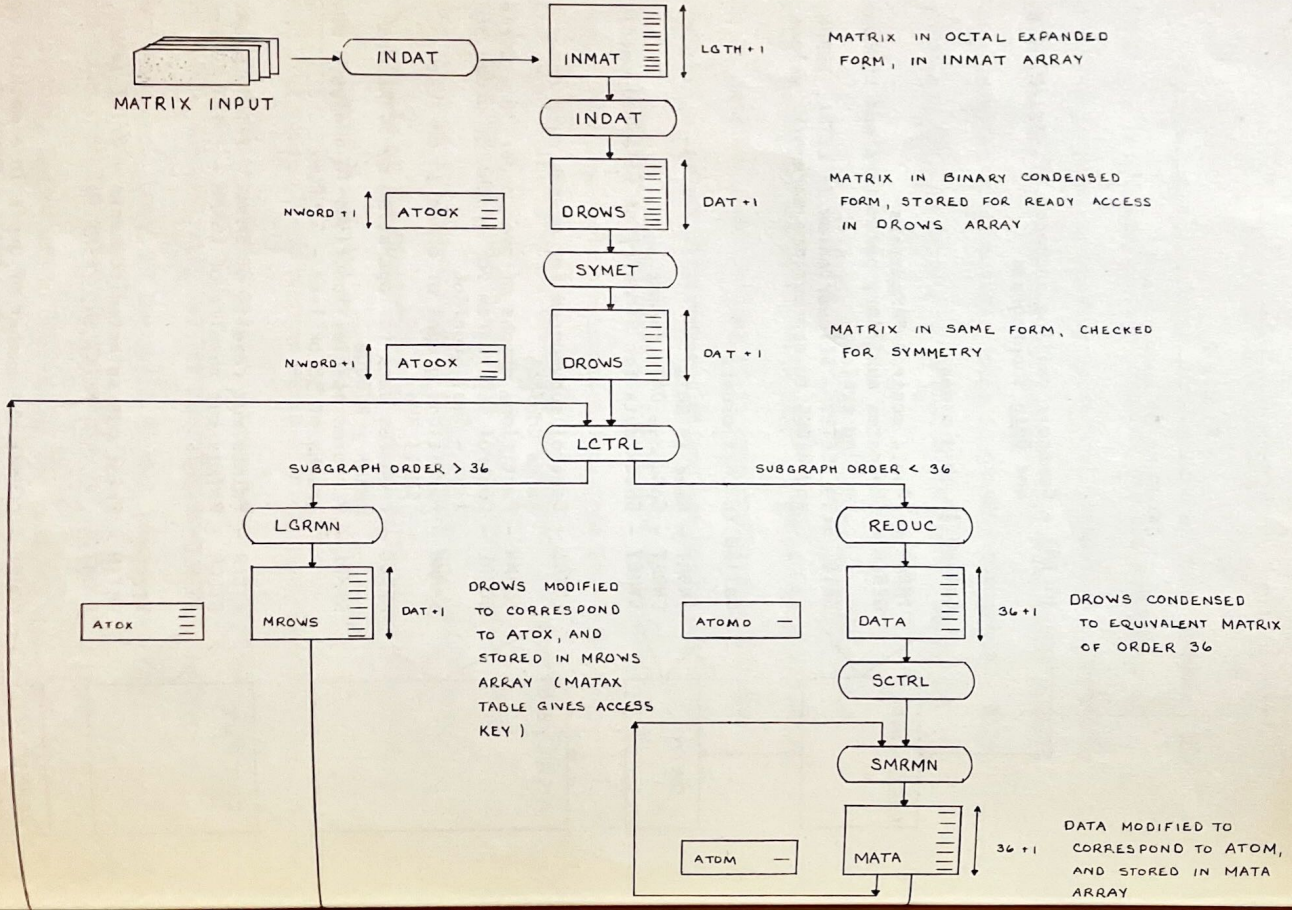
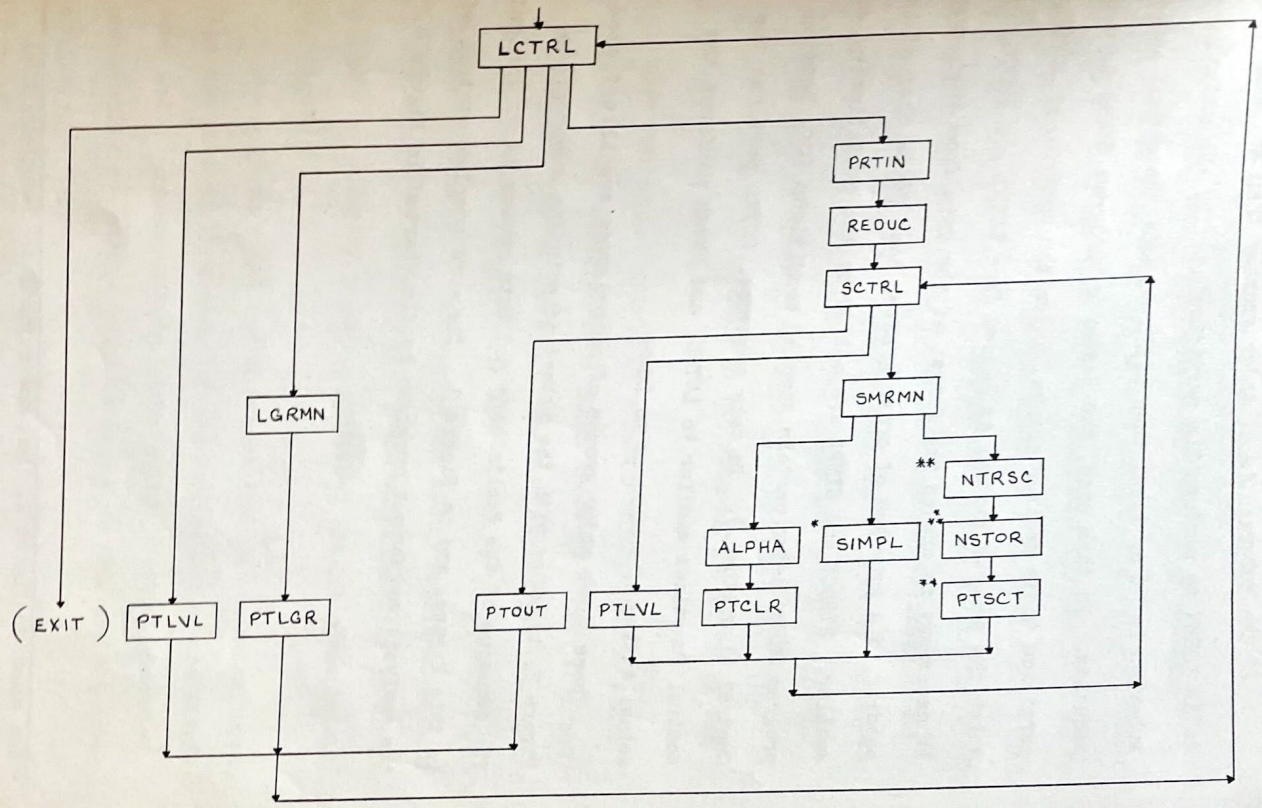


FIGURE 4

THE ANALYSIS SUBPROGRAMS

(ENTRY FROM DATA SUBPROGRAMS)



* = ENTRY POINT IN SCTRL

** = NOT INCORPORATED IN VERSION OF PROGRAM OPERATIONAL IN DECEMBER, 1961

If the subgraph is of order greater than 36, then LCTRL calls LGRMN to perform the partitioning. If the order of the subgraph is 36 or less, then LCTRL selects the second analysis sequence. In this case, the faster subprogram SMRMN is used to partition these smaller graphs, under the control of SCTRL, to which the program control is passed from LCTRL via REDUC. REDUC is necessary to change the format of the data from that corresponding to a subgraph of order greater than 36 to that of one smaller. NTRSC* and SIMPL are called by the partitioning subprogram SMRMN under certain special conditions (Cf. Section 3, Control algorithms.). In this sequence, SCTRL performs the control functions similar to LCTRL, and SMRMN performs the actual analysis, analogous to LGRMN.

These three major groups of subprograms are illustrated in Figure 2, together with the output or printing subprograms. The sequence of the Preliminary and Data groups is as indicated in this Figure, and in Figure 3. The more complex control of the Analysis and Output programs is flowcharted in Figure 4.

---

* Not shown in Figure 2. Not operational in December, 1961.

B. MACHINE REPRESENTATION

Every graph which contains n vertices is in one-one correspondence with a binary square matrix of order n, in which the i-th row and the i-th column both stand for the i-th vertex of the graph. A one (1) in the ij-th cell of the matrix stands for a link between vertex i and vertex j, and a 0 in the ij-th cell indicates that there is no link between vertices i and j. The principal diagonal of the matrix contains zeros, since no vertex is linked to itself. The links are non-directional, so the matrix is symmetrical about the main diagonal (that is, the entry in the ij-th cell is the same as the entry in the ji-th cell).

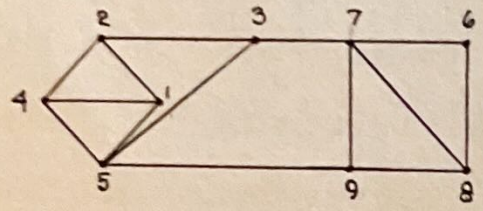
Each row of the matrix is a n-bit binary vector, whose 1's specify the vertices from which there is a link incident on the vertex corresponding to the row. Since the IBM 709 and 7090 have a word length of 36 bits, a single computer word is capable of describing one row of any binary matrix whose order is less than 37. Hence, for a graph which contains 36 vertices or less, an array of 36 words describes the graph completely. Cf. Figure 5.

For a graph which contains more than 36 variables, more than one word is necessary for the description of one matrix row, and a correspondingly larger array, in which blocks of words stand for single rows, is needed to describe the graph completely.

Figure 5

EXAMPLE OF A GRAPH AND ITS MATRIX REPRESENTATION

a) THE GRAPH



b) MATRIX REPRESENTATION

VERTICES:	1	2	3	4	5	6	7	8	9
1	0	1	0	1	1	0	0	0	0
2	1	0	1	1	0	0	0	0	0
3	0	1	0	0	1	0	1	0	0
4	1	1	0	0	1	0	0	0	0
5	1	0	1	1	0	0	0	0	1
6	0	0	0	0	0	0	1	1	0
7	0	0	1	0	0	1	0	1	1
8	0	0	0	0	0	1	1	0	1
9	0	0	0	0	1	0	1	1	0

c) MACHINE REPRESENTATION

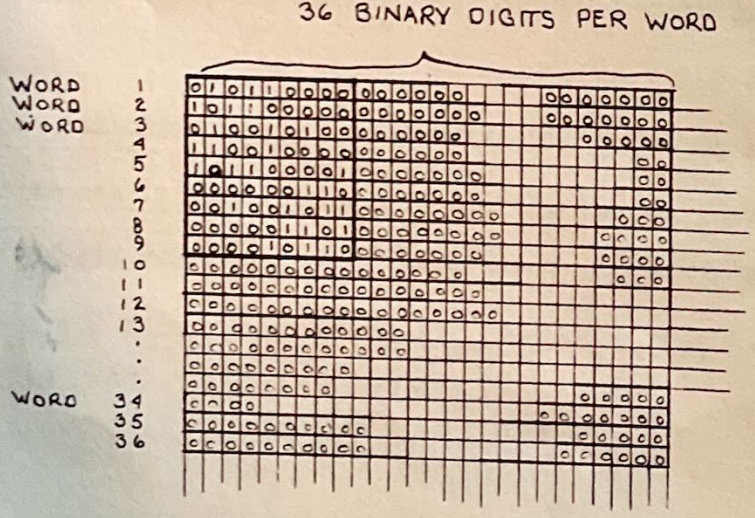
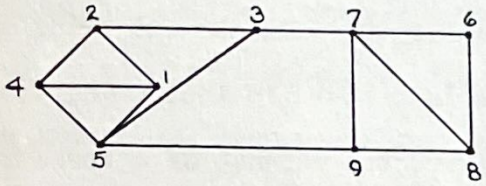


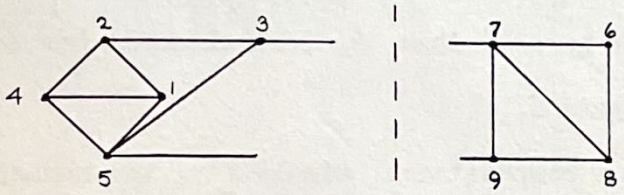
Figure 6

MACHINE REPRESENTATION OF A GRAPH AND ITS SUBGRAPHS

a) THE GRAPH

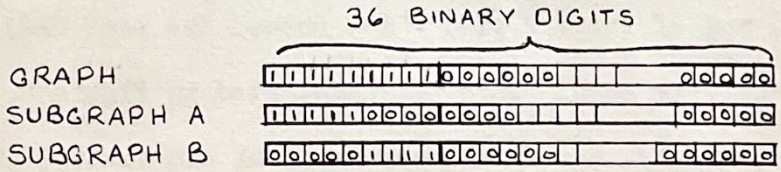


b) A PARTITION OF THE GRAPH INTO TWO SUBGRAPHS



SUBGRAPH A SUBGRAPH B

c) MACHINE REPRESENTATION



NOTE THAT ANY TWO OF THESE THREE WORDS ARE SUFFICIENT TO SPECIFY THE THIRD, SINCE

GRAPH = UNION OF (A, B)

Similarly, when it comes to partitioning the graph, machine words (binary vectors) can be used to describe any given partition. A partition divides the vertex set of a given graph into two or more subsets. If the number of subsets is precisely two (as it is throughout the program*), then the partition is uniquely specified by either of the two subsets, since the other subset is its complement with respect to the set being partitioned. An n-bit binary vector, with 1's indicating the presence of a vertex, and 0's indicating the absence of a vertex, can represent any such set. Again, if the number of vertices in the set being partitioned is not greater than 36, only one machine word is necessary to describe it. If the number of vertices in the set is greater than 36, several words per set are necessary. Cf. Figure 6.

For the sake of computational simplicity, the number of words used is always integral. As a result, the squareness of the matrix means that the matrix storage always contains integral multiples of 36 words. Thus, if the matrix is of order 50, two words per matrix row are used, the last 22 entries in these vectors simply become zeros for the duration of the program, and of the (72).(2) = 144 words, the last (22).(2) = 44 are entirely zero. This is illustrated in Figure 6.

Secondly, again for computational simplicity, and to save computer time, a distinction is made between cases which can

*Except for operations with NTRSC and its associated subprograms. Cf. Section 3, Control algorithms.

be dealt with by a single word (i.e. those where the graph contains 36 vertices or less, so that the order of the vectors required is 36 or less), and those cases which cannot be dealt with by single words. Two different sequences of subprograms are used to deal with these two cases and transfer control back and forth between them, as described above.

Finally, to make the use of index registers and indirect addressing as simple as possible, all sequences of stored information are stored backwards, from their symbolic address.

C. DESCRIPTION OF ANALYSIS ALGORITHMS

The algorithms upon which the analysis subprograms are based can be divided into three groups:

1. criterion- the computation of the measure by which the "strength" of a partition is evaluated
2. sampling- the selection of possible partitions to be evaluated
3. control- allocation and record-keeping with regard to storage of partition results; decisions about sequence of partitioning, when to stop partitioning, and printing out of results.

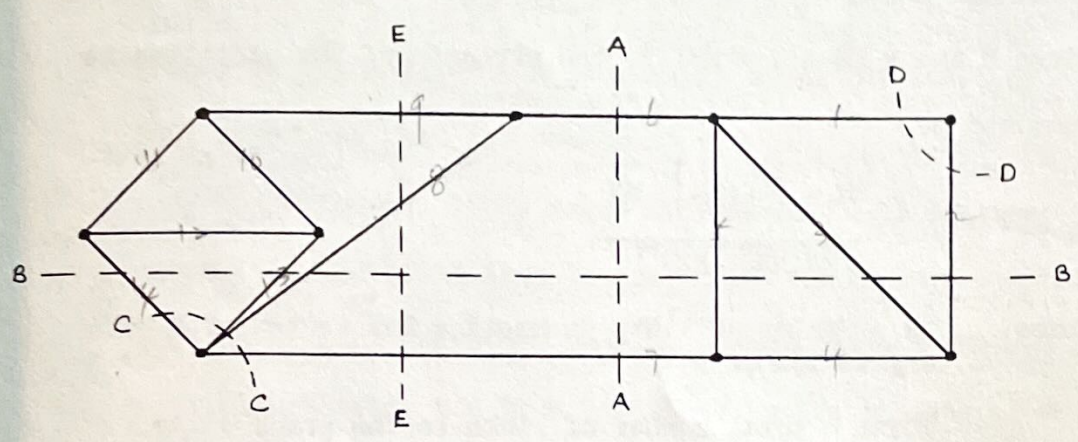
1. The criterion for selecting an optimal partition

The particular criterion upon which this program is based is derived from information - theoretic considerations. The purpose of the criterion is to select one particular decomposition of a graph as best. To discuss the particular criterion, it is necessary to define the basis upon which links are determined.

We assume that each link of the graph represents a statistical correlation between the variables associated with its endpoints (end-vertices). It follows, from considerations of information theory, that the information transmitted from one subset to the other can be used as a criterion for an optimal partition. As shown in Appendix E, it is desired to obtain a partition of the graph's vertex set into two sets which have the least possible information transmitted across the partition.

Figure 7

EXAMPLES OF INFO



NUMBER OF NODES = ORDER = 9

NSQ1 = (9)(8)(1/2) = 36

TOTAL = 19

PARTITION	NO. OF LINKS, RR	NO. OF VERTICES ON EACH SIDE OF PARTITION M	N	INFO	RANKING 1 = BEST
AA	2	5	4	-.104	1
BB	6	3	6	-.003	4
CC	4	1	8	+.004	5
DD	2	1	8	-.006	3
EE	3	4	5	-.071	2

$$STR^2 = \frac{RR - \left(\frac{TOTAL}{NSQ1}\right) MN}{\sqrt{MN(NSQ1 - MN)}}$$

$$INFO = (STR)(|STR|)$$

$$= \frac{2 - \left(\frac{14}{36}\right) 20 \cdot \left| 2 - \frac{14}{36} \cdot 20 \right|}{20(36-20)}$$

$$= \frac{2 - \frac{(14)}{36\,18\,9}\,\overset{7}{20}\,\overset{10}{}}{\sqrt{20(36-20)}} = \frac{2 - \frac{70}{9}}{\sqrt{280}}$$

$$= \frac{2 - 7.78}{16.7} = \frac{-5.78}{16.7}$$

$$= -0.0346$$

$$NSQ1 = \frac{n(n-1)}{2} = \frac{9 \times 8}{2} = 36$$

The mathematical expression of this information, and its normalization, as discussed in Appendix E, lead to the following specific measure of the "strength" of any partition: for a partition which divides the set of vertices into two subsets of sizes M and N (M + N = NBIT), the strength of the partition is measured by

$$STR = \frac{RR - \left(\frac{TOTAL}{NSQ1}\right) MN}{\sqrt{MN(NSQ1-MN)}}$$

where: RR = number of links connecting any vertex of M with any vertex of N

TOTAL = total number of links in the graph

NSQ1 = maximum possible number of links in the graph $\left[\frac{NBIT\ (NBIT-1)}{2}\right]$ $\frac{(n)(n-1)}{2}$, n = no of nodes

MN = (M) x (N).

For computational purposes, the actual measure used is INFO, a monotonic function of STR: INFO is the square of STR, but with the sign of STR preserved. Cf. Figure 7.

The program's central algorithm searches for that partition of a graph's vertex set for which INFO is algebraically minimal.*

2. The selection of trial partitions

As shown above, for a given vertex set, a partition is uniquely determined by giving one of its component subsets, since the other subset is always the complement of the first, with respect to the vertex set under consideration. Let INFO be defined for a given subset, as that value of INFO defined for the partition which this subset determines. Then, the task of finding a

*INFO may be negative, and usually is for minimal points.

partition for which INFO is minimum, is the same as the task of finding a subset for which INFO is minimum. However, the number of possible partitions of a vertex set, being $\frac{1}{2}$ the total number of subsets, is, in the case of n vertices, $\frac{1}{2}.2^n = 2^{n-1}$. For graphs of any interest, n is usually large (at least of order 100, say)*, so that the number of possible partitions becomes very large indeed. This makes it impossible to examine every possible subset, and then to select precisely that one for which INFO is minimum. Instead we must somehow sample the set of all possible subsets, and then use these sample subsets as starting points in a hill-climb search procedure.

More precisely, sampling produces a starting trial subset, which is then modified iteratively, by one point at a time, in an attempt to find subsets whose INFO is lower. This continues until no modification of the subset by one vertex improves the value of INFO. The algorithm thus has three components: the choice of a starting subset; its modification under the rule that INFO has to improve as we go along; and the termination of the modification, at that point where no improvement is possible.

The 2^n possible subsets of the vertex set of n vertices form what is called a lattice. The arrangement of these subsets in the lattice depends upon the simple notion of adjacency between two subsets. Two subsets are called adjacent if one can be made from the other by adding or subtracting a single vertex from it. This is illustrated in Figure 8, where the adjacent

---

*The analysis of the highway interchange problem used a graph with 112 vertices. Cf. Alexander and Manheim, THE DESIGN OF HIGHWAY INTERCHANGES.

partition for which INFO is minimum, is the same as the task of finding a subset for which INFO is minimum. However, the number of possible partitions of a vertex set, being $\frac{1}{2}$ the total number of subsets, is, in the case of n vertices, $\frac{1}{2}.2^n = 2^{n-1}$. For graphs of any interest, n is usually large (at least of order 100, say)*, so that the number of possible partitions becomes very large indeed. This makes it impossible to examine every possible subset, and then to select precisely that one for which INFO is minimum. Instead we must somehow sample the set of all possible subsets, and then use these sample subsets as starting points in a hill-climb search procedure.

More precisely, sampling produces a starting trial subset, which is then modified iteratively, by one point at a time, in an attempt to find subsets whose INFO is lower. This continues until no modification of the subset by one vertex improves the value of INFO. The algorithm thus has three components: the choice of a starting subset; its modification under the rule that INFO has to improve as we go along; and the termination of the modification, at that point where no improvement is possible.

The 2^n possible subsets of the vertex set of n vertices form what is called a lattice. The arrangement of these subsets in the lattice depends upon the simple notion of adjacency between two subsets. Two subsets are called adjacent if one can be made from the other by adding or subtracting a single vertex from it. This is illustrated in Figure 8, where the adjacent

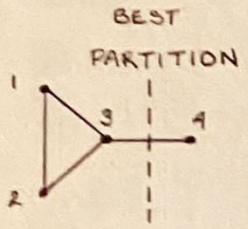
---

*The analysis of the highway interchange problem used a graph with 112 vertices. Cf. Alexander and Manheim, THE DESIGN OF HIGHWAY INTERCHANGES.

Figure 8

EXAMPLE OF A GRAPH AND ITS LATTICE

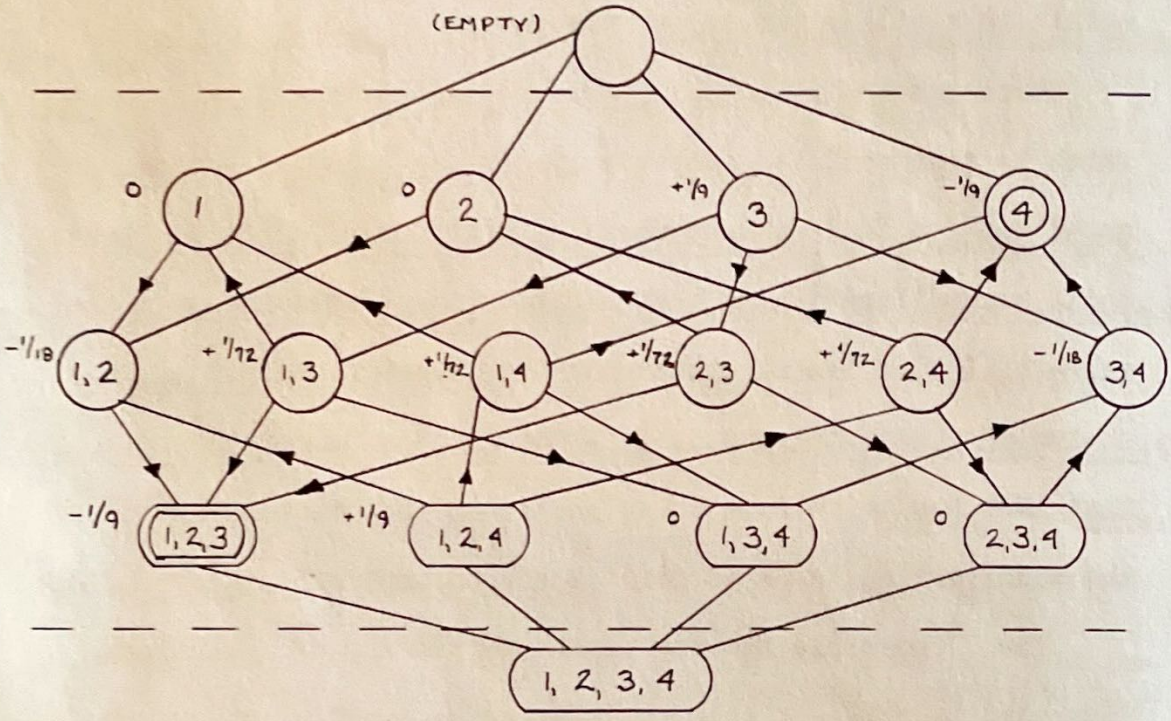
a) GRAPH



b) LATTICE:

(1,2) = SUBGRAPH CONSISTING OF VERTICES 1, 2

= OPTIMAL PARTITION



c) The number beside the subset identification indicates the corresponding value of INFO.
Note that the lattice is the same for all graphs with four vertices. However, the values of INFO (and therefore the arrows) will depend on the specific graph links associated with the four vertices.

subsets are connected by a line. Since every subset has a value of INFO attached to it, we can also associate an arrow with every line in the lattice, showing by its direction which of the two subsets concerned has the lower value of INFO. As a convention, the arrow points toward the subset whose INFO is lower.

The search for better subsets now traces out a path over these lattice lines, always going in the direction of the arrows. There must be subsets which have no arrows leaving them. As soon as the search encounters one of these subsets, it terminates.

There are two points of the lattice which are singular, and must therefore be ignored. These are the full set and the empty set, which both correspond to that imaginary partition that separates the entire set from nothing. Clearly this partition is of no interest. This is expressed mathematically by the fact that for these two subsets, INFO is indeterminate, being 0/0. Arrows cannot be associated with any lattice lines connected to these points; in the program the hill-climbing procedure ignores them.

For the sake of simplicity, we may introduce an analogy, in which subsets are the points of a surface. The altitude of the surface at any point has the value of the INFO of the corresponding subset, and the arrows are always pointing downhill. In this case, the search is equivalent to dropping a ball on the surface, and watching to see where it rolls to.

The analogy makes it clear how critical the choice of starting points can be. The purpose of the search is to find

the very lowest point of the surface. It may well be, however, that the ball finds its way into some valley, not the lowest, but cannot get out again. This is the problem of local minima which occurs in all hill-climbing methods.

In other words the assumption underlying the hill-climbing procedure is that the surface is relatively smooth: that is, the minima whose values are low have correspondingly large "drainage basins," and will therefore be reached from a large number of other points on the surface, while minima whose values are relatively high and undesirable, have relatively small drainage basins. This assumption implies that the best minimum will actually be reached by at least one path, even if the number of starting points is rather small.

There is a difficulty, however. For this procedure to work, the starting points should be equally spaced over the surface. Unfortunately, there is no obvious way of finding points which are equidistantly distributed over a lattice. Finding such a collection of points is equivalent to finding a collection of corners of an n-dimensional cube which are evenly spaced, for edge distance, over the cube. This is a very difficult problem which we have not attempted to solve. Instead, a randomly generated vector is used to select the starting points.*

---

*The selection of an actual starting point is achieved by taking a given random set of vertices, represented by the octal words RANDM, and adding, each time a new sample starting point is to be generated, another random word, DIFF. The resultant random word(s) are then tested to select those vertices which are in the graph, such that those vertices of the graph which are also in the generated random word(s) become the elements of the starting partition.

The trial subset so generated is called TSET. The path-finding component of the algorithm proceeds by testing each vertex which is not in TSET, adding it to TSET and determining if the set so found would yield a lower INFO. If not, then another vertex is selected and tested for addition. If the tentative modification does achieve a lower INFO, that modification is stored.* All the vertices not already in TSET are tested for addition in this manner, one by one.

Retaining the same TSET, each vertex included in TSET is tested to determine if removing that vertex from TSET would result in a better partition.** The best partition discovered in this subtraction loop is then compared with the best discovered in the addition loop, and the better of these two is compared with the partition represented by TSET. If the addition or subtraction of one vertex results in a partition with a smaller value of the corresponding INFO, that partition replaces TSET, and the procedure is repeated. In this manner, a path through the lattice of possible partitions is traced out by additions or subtractions of one vertex at a time.

---

*Since the addition test selects the vertices in the numerical order of their labels, i.e., 1,2,3,4,... this procedure results in a slight bias towards the lowest-numbered vertices.

**The vertices are considered in order of ascending numerical label in the subtract loop, so that here too there is a bias toward lowest-numbered vertices. Furthermore, a partition found in the subtract loop must be better than one found in the add loop, not just equal, in order to replace it. Therefore, an additional bias exists in favor of partitions which add vertices, at the expense of partitions of equal strength but which are formed from TSET by subtracting vertices.