AD 636251

COMPUTER COMMAND AND CONTROL COMPANY

A DATA STRUCTURE FOR DIRECTED GRAPHS IN MAN-MACHINE PROCESSING

OFFICE OF NAVAL RESEARCH
INFORMATION SYSTEMS BRANCH (CODE 437)
DEPARTMENT OF THE NAVY
WASHINGTON, D. C. 20360

COMPUTER COMMAND AND CONTROL COMPANY

WASHINGTON OFFICE - SUITE 1315, 1750 PENNA. AVE., N. W., WASH., D. C. 20006 • (202) 298-6507

A DATA STRUCTURE FOR DIRECTED GRAPHS IN MAN-MACHINE PROCESSING

Contract NOnr 4815(00)

Report No. 77-106-1

Submitted to

OFFICE OF NAVAL RESEARCH 437
Information Systems Branch (Code ~~473~~)
Department of the Navy
Washington, D. C. 20360

June 20, 1966

SUITE 510 WILFORD BUILDING, 33RD & ARCH STREETS, PHILADELPHIA, PENNA. 19104 • EV. 2-3421

TABLE OF CONTENTS

TABLE OF CONTENTS (Contd)

Report No. 77-106-1 June 20, 1966

SUMMARY

This report describes research in information processing intended for application to man-machine communication processes. In order to allow a computer user to represent some problems more easily, we would like to let him name and define relations between information entities with which he deals, and then manipulate a large number of such relations stored by the computer. A set of statements on the relative desirability of various conditions is an example of a set of relations which the user might want to store and manipulate.

The problem is that available representations for such sets of relations tend to be unwieldy and may require a great deal of processing for large numbers of relations. In order to make such a capability available, compact and easily-manipulated internal computer representations for large numbers of the various kinds of two-entity relations must be found. This report describes such a development for one basic logical form of relation, the transitive, anticommutative relation exemplified by "precedes", "includes", "is greater than", and similar phrases.

The report describes a method for storing directed graphs (of transitive anticommutative relations) in a list-structured computer memory. There are three important features of this representation method: A method of dividing a graph into a number of strata based on the lengths of paths in the graph, a recursive decomposition technique which prod es successively less complex versions of the graph, and a recursive sea technique which utilizes the stratification and decomposition to extract information from the graph with a limited amount of processing effort. Some preliminary tests of the representation technique on graphs containing up to several hundred randomly chosen relations are described; the results of the test indicate that this representation may require less processing time and far less core storage than previously-used techniques when applied to large graphs.

Report No. 77-106-1 June 20, 1968

A DATA STRUCTURE FOR DIRECTED GRAPHS IN MAN-MACHINE PROCESSING

1. INTRODUCTION.

There is a wide variety of methods of data storage for problems which can be defined before they are programmed. For ill-defined problems in which the form and scope of the problem may be defined as the result of a joint man-machine process (long after most of the computer programming is complete) few methods are available for construction and use of an appropriate data structure ad hoc. The most effective procedures for working with incompletely defined data structure are those associated with list processing languages such as LISP and IPL-V. While these are powerful tools, they tend to be rather clumsy and/or inefficient in memory for representing some data relationships which occur repeatedly in man-machine communication and in the problems which are represented by such communication.

The process of man-machine communication is greatly aided when the machine is able to deal directly with the symbols and relationships which the man uses to represent his problem. The most restrictive limitations of available methods seem to be associated with the man-to-machine communication rather than the machine-to-man link or internal processes. Much of the man's representation of problems is unacceptable to machines because of its vagueness, incompleteness, ambiguity, nonformality and the like. We must find ways for the machine to freely accept and use incomplete, qualitative information, to mix that information with internally-derived information, and to accept modifications as easily as the original information is accepted.

Two areas of interest in which such a capability is needed are the modeling of natural language by machine and the acquisition or modeling of complex human decision processes by machine. In the case of natural language, people use the language not only to express information relevant to problems but to define, restrict and explain their use of language as well. Thus, language models must change in response to speech; open-ended data structures are therefore required to accommodate such models.*

* Current linguistic theory does not accommodate such models, although the need for them is recognized.

For modeling of decision processes, such an open-ended capability will be useful in cumulative construction of process models which could not be handled by the usual process of description, programming, debugging and operation because of the difficulty (on the man's part) of conceiving of the process as an organized whole. By developing a process model keyed to the man's conceptions (including relations between them) in an experimental way, the machine may overcome the man's conceptual limitations by its different way of using memory, and thereby break through the limits which prevented programming the decision process in the first place.

The effort described in this paper is an attempt to select a recurrent relationship of data structure, to represent the relationship for efficient open-ended storage in computer memory and to find ways to manipulate such a representation easily for large amounts of data.

The relationship selected is the transitive, non-commutative relation between two objects (symbols), represented in various applications by linear inequalities, set inclusion, time precedence, priorities, and in many other ways. The transitive property simply means that if

A (relation) B and
B (relation) C

are true, then

A (relation) C

must be true; the non-commutative property simply means that if

A (relation) B

is true, then

B (relation) A

is false. We will symbolize all such individual relations below by $<$.

A collection of relation statements such as those above concerning a set of symbols can be conveniently represented by a directed graph (see Fig. 1-1) in which each line represents one explicit relation.

Relations

$A < B$

$B < C$

$C < E$

$A < D$

$D < E$

$F < D$

Equivalent Graph

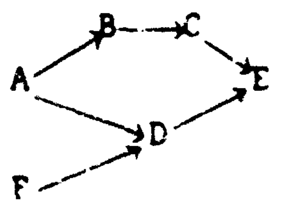


FIGURE 1-1 - REPRESENTATION OF RELATION STATEMENTS BY A DIRECTED GRAPH.

The case of large graphs is of more interest than the small graph case, since the latter can often be handled most effectively by simple exhaustive search or storage techniques. Graphs having hundreds or thousands of vertices occur in models of process precedence (as in PERT charts), organizational relationships, various natural and artificial language models, assembly-sub-assembly relations, state and subgoal dominance in games and the like.

The usefulness of such relations has been demonstrated by Bertram Raphael, among others, in his work on Semantic Information Retrieval.[1] In providing means for computer response to English statements and questions, Raphael kept eight sets of relations between objects or sets in the computer memory. They were designated as "set-inclusion, set-membership, equivalence, ownership-general, ownership-specific, part:whole general, part:whole-specific, left-to-right position. Most of these relations can be fully represented by graphs of transitive, non-commutative relations with some annotation concerning the objects so represented. It is noteworthy that Raphael considered his system "cramped for memory space" although successful in demonstrating the power and flexibility of the approach.

2. PROBLEM STATEMENT.

The problem is to find a representation for large transitive non-commutative graphs of unrestricted topology, and to find a set of algorithms which will, in combination, use small amounts of memory to store this information, and to enter and extract information rapidly. This problem differs significantly from problems of representing and searching a graph which is implicit in certain information already at hand as, for instance, the move tree in a chess game which is implicit in the present position and the rules of chess.

The above statement is too vague to support evaluation, and so some clarification is required. Performance of any actual programming system will depend on the specific computer, and on:

1) the amounts and types of usable memory,
2) the relative frequencies of the graph manipulations used,
3) the "typical" form of the graphs used.

The memory considered here is essentially the computer's fast memory, normally core. However, it will be seen that the techniques presented have advantageous natural extensions to slower memory and to paged memory. More specifically, we will examine what can be done with 24K words of data space, tacitly assuming a 32K machine with 8K of executive and programs. This provides a convenient base for scaling to other systems.

The potential number of relations which might be stored in a graph rises rapidly with the number of vertices, N. It is $\binom{N}{2}$, or approximately $N^2/_2$; a graph of 1500 vertices with all relations explicit would surely exceed our allotted core. However, nearly all these relations are redundant, being implied by other relations and transitivity; only 1499 explicit relations are needed to imply all of the others. More generally, the elimination of redundant relations from storage offers significant reductions in memory requirements at the expense of some increase in the computation required for manipulation.

As a measure of the storage space required by a graph, we use a combination factor depending on both the number of vertices and the number of arcs and, of course, the amount of information stored concerning each.

A rich variety of graph manipulation processes should be available to the user. However, for reasons which will become obvious later, the process of searching a graph for the presence or absence of a specific stored relation can be used as a representative manipulation process. It is required in both entry and query processes and can be the major time consumer in manipulating large graphs.

A measure of the processing requirements of specific conditions (of graph form and representation) may be considered to be the product of two factors:

1) the time required to traverse a single arc in the graph and decide upon the outcome of that traverse, and

2) the number of arcs are traversed in the search.

It is the latter term which is subject to combinatorial explosion; and so considerable increases in the time required to traverse an arc are acceptable if they serve to limit the number of arcs traversed. These terms are both dependent on the algorithm of search; for a given graph representation several search algorithms are possible. For the evaluations in this report a recursive search algorithm was used; it is described in Sect. 3.3.

The first term is dependent on the computer used, the general abilities of the list processing software and the cleverness of the specific graph manipulation programs The second term is independent of these factors and depends only on the information actually processed and the possible techniques of search limitation inherent in a representation. The more general comparisons therefore depend on the second term alone. For this reason we take the number of arcs traversed as the measure of search effort in this report.

The form of the graphs encountered will depend strongly on what is being modeled, the graphs being partial or complete representations of problem entities. For the purposes of this development we may use random graphs in place of problem model graphs to evaluate approaches, recognizing that a system which works for a variety of random graphs may not be effective in some restricted and improbable class of problem structures. On the other hand, an approach which is ineffective with random graphs may be expected to have very limited utility as a general purpose tool.

The random graphs used for evaluation here are made up of arcs which are selected from all of the possible arcs consistent with an arbitrary total order of the vertices of the graph, all such arcs being equally likely. It is important to note that neither the existence of an underlying total order of the vertices nor the equal likelihood of the arcs are essential to the representation technique developed. Many other kinds of random graphs are possible; the cases in which the underlying structure is not total order and in which the arcs are unequally likely are of definite interest but have not yet been considered in depth.

3. TECHNICAL APPROACH.

3.1 Memory Structure.

One of the first problems to be faced in choosing a computer representation is the choice of a style of memory representation. Since we are particularly interested in maintaining an open-ended structure which is easy to expand and yet efficient, some form of dynamic memory allocation is required. We have chosen a word-oriented list-structured memory of the sort used in IPL-V and LAP.

In a list-structured memory, each cell of storage may be linked to another cell without regard to its address, thus forming lists of cells of storage. These links may be altered by the programs using the memory. Names of lists may be cited freely in other lists or in the list itself, thus allowing extremely complex information organization. The ability to add links to additional cells as those cells are needed allows programs to operate without fixed or preplanned memory allocation, acquiring and returning cells to a public "list of available space" in response to the progress of the computation. For introductory information on list processing, see Reference 2.

An alternative to such a structure is a packed-word matrix representation in which there is a row and a column for each vertex of the graph and each intersection of a row and column is represented by one computer bit. For very large graphs the requirement that all the possible relations be represented tends to outweigh the advantages of single bit storage relations. The list-structured memory approach requires storage only for the relations actually present in the graph rather than all possible relations. The matrix approach is relatively inflexible, but may be advantageous for some kinds of problem structure. It is of particular interest with respect to the recursive decomposition technique described in this report as a method for representing subgraphs of the decomposition.

The list memory may be used to represent the vertices and arcs of the graphs as follows. Each vertex of the graph to be stored is associated with a list structure in memory, with the list structure for a vertex having sublists of the precedent and subsequent vertices in the graph. These sublists represent the arcs. If each vertex list structure contains citation of the precedent and subsequent arcs, then each arc is, in effect, represented twice, allowing search in either the precedent or the subsequent direction.

Information associated directly with the vertex itself may be placed in the main list for the vertex rather than its sublists. Such information might include the external name of the vertex, and bookkeeping information always associated with a vertex. For information which is associated with only some of the vertices, it is more convenient and compact to use the "description list" capability found in nearly all list processing systems.*

3.2 Storage Efficiency.

The efficient storage of information in directed graphs is one of the principal objectives of this development.

What constitutes storage efficiency for directed graphs? How may it be measured? It is conceivable that a measure of the information stored in transitive, anticommutative acyclic directed graphs could be developed based on an enumeration of them, and assignment of their probabilities of occurrence. This would yield a number of bits necessary to identify any particular graph which might occur, and then a reference for evaluation of any other storage scheme. However, the combinational mathematics necessary for such an evaluation, or for the enumeration of the class of graphs is apparently not yet available. (Such mathematical developments would also be valuable in statistical inference problems, where the mathematical form of a relation is to be inferred from a graph of randomly selected examples of the relation.) It is therefore necessary arbitrarily to choose some other reference for measuring the storage requirements imposed by a particular implementation scheme.

* A special list is associated with the list for the vertex in a way which assures that it will normally be ignored in processing. This list contains an arbitrary number of pairs of cells, each pair containing an "attribute" in the first cell and a "value" of that attribute in the second. The attribute may be used to identify the role of the information in the value cell. These lists are created and maintained automatically by the list processing system, and are often referenced by use of a separate set of commands.

The reference chosen for this study is a representation of the graph in which every relationship implicit in the graph is explicitly stored. More specifically, a list-structure representation of such a graph is postulated. The form of the structure is identical with the structure developed in this report except that the "overhead" of storage required for the information included for aiding search processes would be eliminated. The remaining structure consists of a list for each vertex naming a sublist of precedent vertices and a sublist of subsequent vertices, and the two sublists themselves, citing explicitly all precedent and subsequent arcs. This appears to be the form which is most closely comparable to the structures developed in this study. Its storage requirements are directly comparable, and the comparison may be projected to other systems relatively straightforwardly.

For further reference such a graph will be called a "fully explicit" graph and its list structure representation a "fully explicit representation" (see Fig. 3-1b). In contrast, the graph in which there are no arcs representing relationships which are derivable from the transitivity property applied to arcs which are shown, will be called the "basis graph" (see Fig. 3-1a). Arcs which appear in the fully explicit graph but not in the equivalent basis graph are called redundant arcs. It can be shown that the basis graph is unique.

There is a further reason for using the fully explicit graph as a comparator. It is in many ways the simplest representation to understand and to program for. It has been used in studies in which efficient representation was not an object, and it works moderately well for small graphs and for graphs with low information content (see Reference 1). It is thus a feasible alternative in many applications.

The approach of this development may be viewed as, first, gaining of storage efficiency by removing redundant arcs, and second, addition of information to the resulting representation to make search and change of the resulting graph efficient.

In the first part, the reduction in storage requirements is a direct function of the number of redundant arcs which can be represented without explicit storage by the arcs of the basis graph. This is a topological, machine-

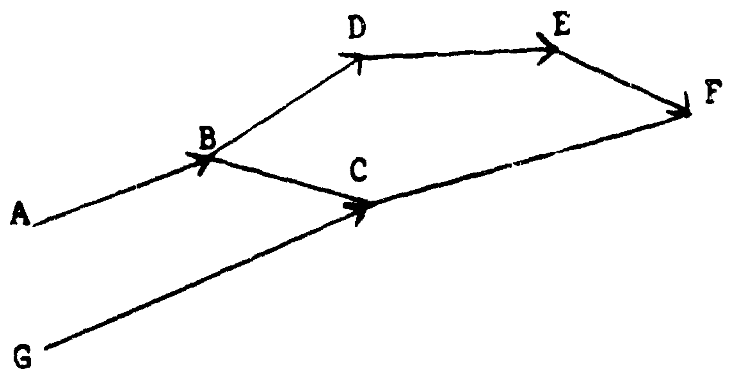


FIGURE 3-1a - BASIS GRAPH.

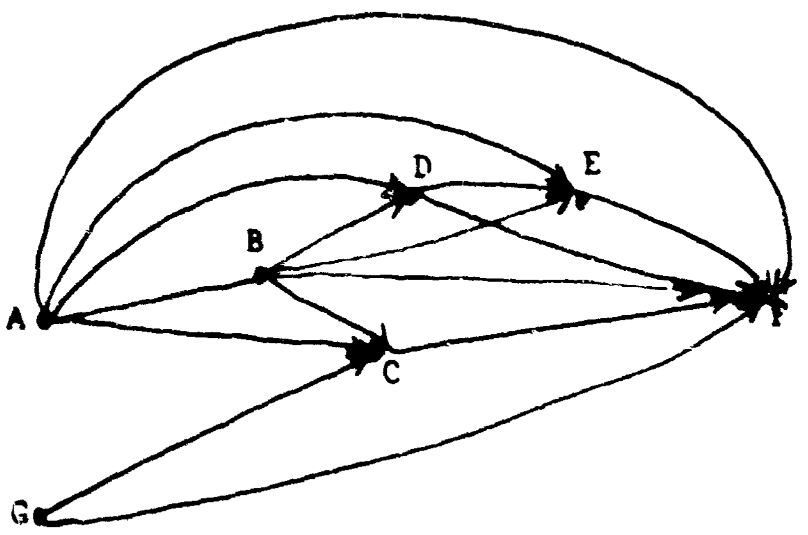


FIGURE 3-1b - "FULLY EXPLICIT GRAPH" SHOWING ALL IMPLIED RELATIONS (REDUNDANT ARCS).

independent property of the graph to be stored. Some estimate of the potential saving may be made from the limits of the number as a function of the number of vertices and information content of the graph. For the empty graph of N vertices and no arcs, there is no difference in the number of arcs to be stored. For the case of a totally-ordered set of N vertices, the basis graph has N-1 arcs and the fully explicit graph has $\frac{N(N-1)}{2}$ arcs, requiring storage of only $\frac{2}{N}$ as many arcs in the basis graph. For graphs which partially order the vertices, the proportion of arcs which are in the basis graph is often (perhaps always) in the range 1 to 2/N. This ratio is one of the parameters which characterizes a graph.

The number of arcs in graph G is denoted as A (G) with the number of arcs in the corresponding fully explicit graph and basis graph as A (G_E) and A (G_B) respectively.

A second aspect of a graph's structure concerns the information content of the graph. For any graph the ratio of the number of arcs represented by that graph (i.e., A (G_E)) to the maximum possible number of arcs representable on the same set of vertices, $\frac{N(N-1)}{2}$, is a measure of the graph's information content. We define a "fullness ratio"

$$F(G) = \frac{2A(G_E)}{N(N-1)}$$

We should note that the fullness ratio is also the probability that for a randomly selected pair of vertices (with all pairs of two different vertices equiprobable) there is an arc between the two vertices represented in the graph. This is the basis for a sampling procedure for estimating F (G) which is described below in connection with the results of Monte Carlo experiments.

3.3 Search as a Representative Graph Manipulation Procedure.

The purpose of this section is to define "search" of a graph in a particular way and to show in what way it is a representative process for a larger range of processes for development and use of graphs.

In order to have a basis on which to judge the computation requirements of any particular graph representation and manipulation algorithms, we made a list of operations which might be made available in a language for man-to-machine communication in problems involving directed graphs. It is not defined as a language, nor is it extensive enough to be useful alone. However, it is intended to cover most of the basic or primitive manipulations on which a more adequate man-to-machine communication language (for this logical form of relations) could be based. It is presented here only to show the extensive involvement of a basic "search" operation in other processes. Each entry in Table 3-1 is in the form of an example of a desirable statement.

The processes marked "X" have as a basic part the determination of the relationship, if any, between two vertices. The test for the existence of a relation is a necessary preliminary process or included process for displaying paths in the graph which imply the relation. It is necessary in the entry and deletion processes in order to be able to reject commands which with the graph, deny the assumptions concerning the relationship being graphed.* Because the test for the existence of a relation is so pervasive in this sample of possible primitive processes, it appears to be a good representative for objective judgements of the overall processing effort which might be required by a particular graph representation. It should be pointed out that no one process or combination of processes can accurately reflect what will happen when a system of algorithms is applied to a particular problem structure. We expect that the effectiveness of the algorithms will be strongly dependent on problem structure in some cases.

The test for a relationship between two vertices, which we call search here, may be performed by techniques which differ in several important ways. The technique used for a comparator here may be described generally as follows:

1) An origin of search is chosen at one of the vertices.

2) One arc is traversed from the origin vertex.

* The three deletion processes are included because, although correction and change of the graph are necessary, some kinds of change would require a great deal of processing and so should be avoided where possible. These processes will be defined formally in a later report when a more complete language is presented.

TABLE 3-1

LIST OF OPERATIONS

Relation Declaration

"precedes" = Transitive Anticommutative Relation

Information Insertion

A precedes B	X

Information Deletion

Deny A precedes B	X
Expunge A precedes B	X
Eliminate A precedes B	

Queries

Does A precede B?	X
Does A immediately precede B?	
Could A precede B?	X
What is the "precedes" relation for A, B?	X
What are the immediate precedents of A?	
What does A immediately precede?	
What is one set of relations which relate A and B?	X
What are all the relations between A and B?	X
What is the maximum number of relations separating A and B?	X
What events related A and B?	X

3) Each applicable "termination condition" is tested for. Some termination conditions include:

a) Destination vertex found.
b) No more arcs to traverse from that vertex.
c) Vertex reached has been reached in this search before.

(There are other termination conditions for graphs having path number stratification; the technique is somewhat elaborated for search after decomposition.)

4) If no termination condition is detected, the vertex reached is used as an origin and the process is repeated recursively. If the destination vertex is found, no more arcs are traversed; the result is reported. If some other termination condition is found, another arc from the present origin vertex is traversed. (If there are no more, this becomes a termination condition for the previous recursion.)

In order to evaluate search effort in machine-independent terms, the total number of arcs traversed is used as a measure.

3.4 Path Number Stratification.

3.4.1 Concept.

The purpose of this section is to describe the concept of path number stratification of a directed graph and to show how it can be used to reduce search effort. In order to describe the process we introduce the following notation:

A graph G (V, A) consists of a set of vertices V and a set of arcs A. The vertices are identified $v_1, v_2, \dots v_i \dots v_N$, where N is the total number of vertices. (We consider only finite N.) The arcs are identified as a_{ij}, where a_{ij} corresponds to an ordered pair of vertices v_i, v_j in which v_i precedes v_j and $v_i, v_j \in V$; then v_i is called the precedent vertex and v_j is called the subsequent vertex. If $a_{ij} \in A$ and if there is no P_{ij} other than a_{ij} in G, then v_i is called an immediate precedent for v_j, and v_j is called an immediate subsequent of v_i. An integer associated with a vertex v_i is denoted S(i), F(i)...etc., according to its function. A path P^x_{it} is an ordered set of adjacent arcs $a \in A$, $a_{ij}, a_{jk} \dots a_{st}$ where x is an arbitrary index denoting the x^{th} path according to some ad hoc definition. If P_{ij} is in G, then v_i is called a precedent of v_j and v_j is called a subsequent of v_i; we write $v_i < v_j$,

$v_j > v_i$. Arcs a_{ij} and a_{km} are adjacent in the order a_{ij}, a_{km} if $j = k$. By convention, there is never more than one a_{ij} for a given i, j. The anticommutivity property is the property that the graph contains no paths P_{ii}. The transitivity property means that the graph G associated with relation R represents all relational statements of the form v_i R v_j for which $P_{ij} \epsilon$ G (rather than just those for which $a_{ij} \epsilon$ A).

Every directed graph of a set of transitive, anticommutative relations has one or more vertices which have no precedents and/or no subsequents. Considering for the moment only the set V_p of vertices having no precedents, several properties of the graph can be demonstrated immediately:

1) There are no paths P_{ij} such that v_i, $v_j \epsilon V_p$.

2) For every vertex $v_j \not\epsilon V_p$ there is at least one P_{ij} having $v_i \epsilon V_p$.

3) All paths in the graph are finite.

We may associate an integer S(i), called the "path number" or "Stratum number" with each vertex v_i according to the following rules:

1) For $v_i \epsilon V_p$, S(i) = 0

2) For $v_i \not\epsilon V_p$, identify all paths P^X_{ji} such that $v_j \epsilon V_p$. Of these identify one containing the largest number of arcs; let this number of arcs be B (P^X_{ji}). Then S(i) = B (P^X_{ji}).

3) Let V_s = (all v_i having S(i) = s). V_s is called the s th stratum of G.

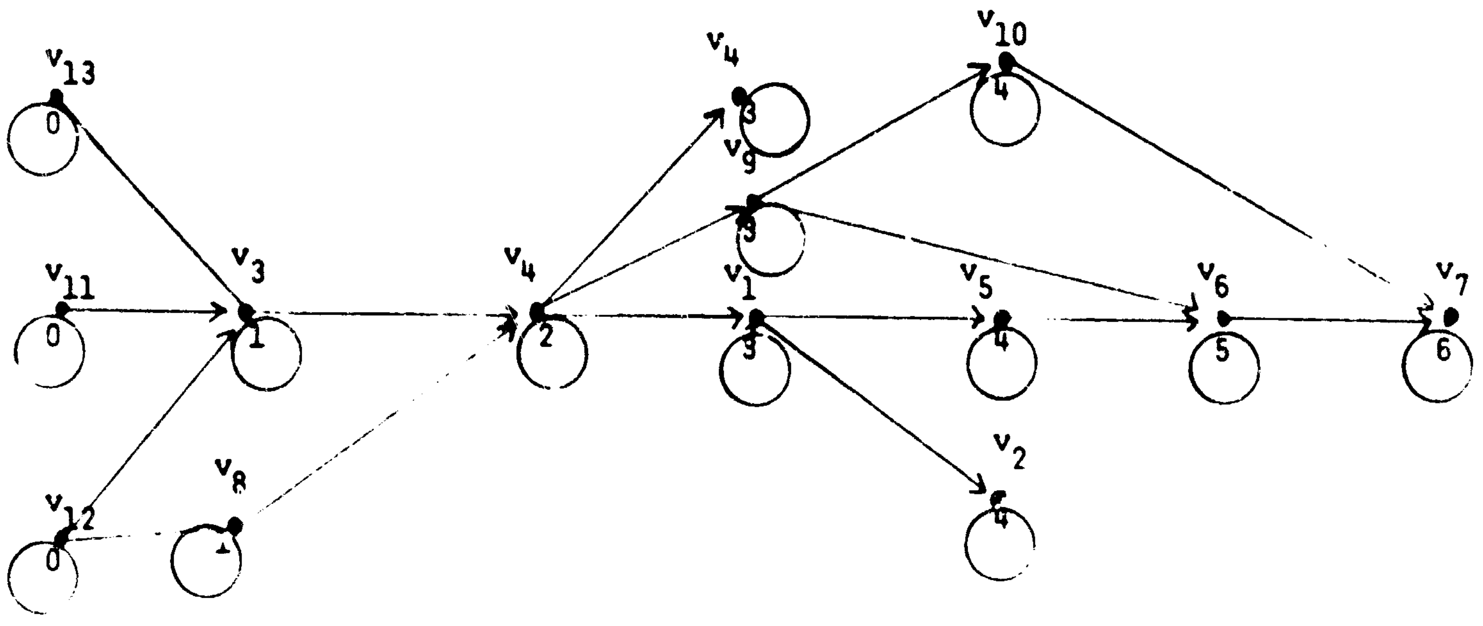
Figure 3-2 shows a graph which has been marked with path numbers.

The following properties of a stratified transitive anticommutative graph can be demonstrated:

1) All vertices are assigned to strata.

2) Path numbers used are consecutive.

3) The path numbers are not changed by addition or deletion of redundant arcs.



Path numbers: S

FIGURE 3-2 - SAMPLE GRAPH WITH PATH NUMBER STRATIFICATION.

4) Any path P^x_{ij} consists entirely of arcs which relate vertices v_k having $S(i) < S(k) < S(j)$.

5) If $S(i) > S(j)$ there is no path P_{ij}.

3.4.2 Search in Stratified Graphs.

We are now in a position to compare search in stratified graphs with search in unstratified graphs, using basis graphs in both instances. Consider first the unstratified graphs, and the cases in which the relationship sought is not found in the graph. In order to complete the search from the chosen origin vertex, every arc which lies on any path which includes that vertex must be traversed at least once, since the only path to the destination vertex might include that arc. The search is performed in two parts, one in the precedent direction and one in the subsequent direction. The final termination condition is in the absence of any more arcs to traverse in the sequent (precedent or subsequent) direction. For graphs in which most pairs of vertices are related, this represents a search of nearly the entire graph, a tedious process for large graphs. Considering searches on relationship which are represented in the graph, such extensive searches will occur part of the time simply because the first arcs traversed lead out of the neighborhood of the destination vertex. An extreme case is illustrated in Fig. 3-3, in which a search for the relationship between v_1 and v_2 traverses the arcs in the order shown, searching from v_1 first in the precedent direction and then in the subsequent direction. On the eleventh arc traversed (out of only 16 in the graph) the relation is discovered.

Consider search for a relation between v_i and v_j in stratified graphs. At the beginning of the search, the path numbers $S(i)$, $S(j)$ are determined. If $S(i) = S(j)$, no further examination is necessary, since there cannot be a path P_{ij}. If $S(i) < S(j)$ then, if v_i is arbitrarily chosen as the origin, search can be confined to the subsequents of v_i, since there can be no path P_{ji}. Upon traversing an arc a_{mk}, if $k \neq j$ the condition $S(k) \geq S(j)$ can be used as a termination condition on the particular recursion of the search process currently in progress. For example, performing the same search cited above in the equivalent stratified graph shown in Fig. 3-2, the arc a_{15} would be traversed

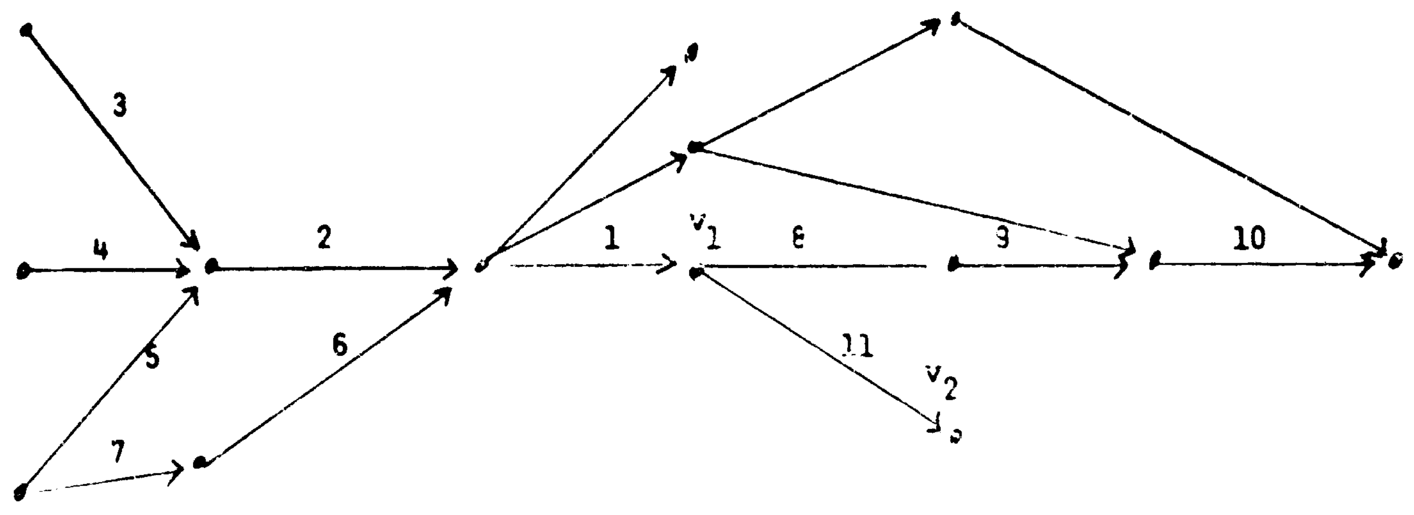


FIGURE 3-3 - SEARCH FOR A RELATION v_1, v_2 IN AN UNSTRATIFIED GRAPH.

first, resulting in a termination condition $S(5) = S(2) = 4$. Then arc a_{12} would be traversed, completing the search on the second arc.

In general, the path number limitation on search confines the region in which search will be continued to the strata between the terminals of the search.

For searches to confirm or deny a specific relationship, the path numbers can be used to deny some relationships without specific examination of the graph. If all possible searches for specific relations are equally likely, then somewhat more than half of the searches can be satisfied in this way; for every pair of vertices v_i, v_j in which $S(i) \neq S(j)$, either search for P_{ij} or search for P_{ji} will be rejected and the other accepted, and if $S(i) = S(j)$ both will be rejected.

The effects of this search limitation techniques are subject to measurement by Monte Carlo experiments, as discussed in the section on experimental results.

Updating of the path numbers associated with each vertex must be performed each time information is added to the graph. A simple recursive procedure can operate on the graph to perform this updating, beginning with the precedent vertex of the newly-added arc. The procedure is similar to the search procedure described above, except that 1) there is no termination condition corresponding to finding the destination vertex, 2) a termination condition for "no change in path number required" is added, 3) changes in vertex path numbers are made as the updating proceeds. The update procedure thus deals with only the portion of the graph requiring updating rather than the entire graph.

The effort of path number updating for large graphs can be significant on those occasions in which the longest path in the graph is being extended. There are at least two techniques which will limit the update effort at some expense in terms of increased search time (because some specific termination conditions on certain searches are eliminated.) The first is to let the path numbers be relative to certain fixed strata in the graph rather than relative only to the stratum of vertices without precedents. If ten fixed strata are introduced into a graph with 1000 arcs on the longest path, then the worst case path number update deals with only about 100 vertices instead of 1000, a 90% saving. Whether there is an overall saving in any useful case in unknown.

A second approach consists in making the early entries to a graph with strata numbered in multiples of some constant Y, rather than consecutively. The later accessions can be made with a reduced value Y2 (for example, $Y_2 = \frac{Y_1}{2}$), thus tending to fill up the empty strata between the original strata and reducing propagation of path number change sharply. The value of this tactic is likewise unknown.

In usage situations in which the proportion of effort spent on entry of information is very small, the increase in search effort would exceed the saving in updating effort. Both of these variations destroy the property of consecutivity of path numbers, a property which is useful in implementing some of the functions in the graph language operations above.

3.4.3 Generalization of the Stratification Concept.

The concept of path number stratification may be generalized as follows. Any set of vertices V_a may be chosen as a basis for stratification, provided that 1) there are no paths P_{ij} or P_{ji} for any v_i, $v_j \epsilon V_a$ and 2) for all $v_k \not\epsilon V_a$ there is a path P_{ik} or P_{ki}, $v_i \epsilon V_a$. All members of V_a are associated with a symbol $S_a(i)$ chosen from an ordered set of symbols R having the property that the order relation between any pair of symbols r_i, $r_j \epsilon R$ can be determined in some sense "by inspection", i.e., without reference to the graph. Then any other set of vertices V_b, may be chosen provided that 1) there are no paths P_{ij} or P_{ji} for v_i, $v_j \epsilon V_b$ and 2) there are no paths of a particular order (either P_{ij} or P_{ji}) for $v_i \epsilon V_a$, $v_j \epsilon V_b$, and 3) for each $v_i \epsilon V_b$ there is at least one path of the admitted order (P_{ji} or P_{ij}) for some $v_j \epsilon V_a$. The vertices of V_b are associated with a symbol $S_b(i)$ from R having an order relation to $S_a(i)$ consistent with the order relation of the paths in G between members of V_a and members of V_b. Another set of vertices V_c may now be chosen under the same restrictions, with the restriction on consistency of order relations applying to each of all sets previously chosen (V_a and V_b) and the choice of $S_c(i)$ likewise limited to consistency with the path directions to previously chosen sets. Sets are chosen until all the vertices are associated with symbols S(i). The order of the symbols S(i) then provides the same termination conditions on search which were described above for path number stratification.

Two approaches to a more general stratification appear particularly attractive. The first is to perform two stratifications instead of one, using the set of vertices without subsequents as the basis for the second stratification; otherwise applying the same algorithm, traversing arcs in the opposite direction. This would provide an independent search termination condition which in some ways would be maximally different from the precedent-based stratification. Path numbers (taken with the same direction of increase) would be maximized rather than minimized subject to local irregularities. Of course, for graphs which are trees or which are constrained to having all paths between any pair of vertices be of equal length, a second set of path numbers would do absolutely no good. A hand Monte Carlo test of the number of arcs traversed in 20 searches, randomly selected in a basis graph of 50 vertices and 95 arcs and a fullness ratio of about .37, had the results tabulated in Table 3-2. These results indicate that, at least for this neighborhood of fullness ratio and this method of information selection (and thereby, degree of irregularity), the second stratification might materially reduce search effort, although the present result is certainly inconclusive. It is interesting that the searches improved were mainly the longer searches.

A second approach would involve maintaining (or periodically identifying) the largest set of pairwise unrelated vertices, and using that set as a base for a stratification. The qualitative result would be that many searches would terminate at that stratum, and that all searches on pairs of vertices within that stratum would be rejected without reference to the graph.

3.4.4 Maintenance of Stratified Graphs.

While the maintenance of the basis graph results in major savings in storage, the processes by which redundancies are removed or prevented may be major users of computer time. The problem is not in preventing the entry of a redundant arc, since this can be done by a simple search as part of the entry process. It is rather in detecting the instances in which entry of a new arc makes a previously-entered arc redundant.

There is no limit on the remoteness (counting along paths or strata) of an arc v_{az} which may be made redundant by entry of an arc v_{ij}. There is no simple inference from the path number updating behavior of an

TABLE 3-2

A MONTE CARLO TEST IN TWENTY SEARCHES

	Count	Average Difference in Rank on Underlying Total Order of the 50 Vertices	Arcs Traversed Using Precedent-based Path Numbers Only	Arcs Traversed Using Both Precedent- and Subsequent-based Path Numbers
Searches Affected by Second Stratification	5	17.8	64	36
Searches Unaffected by Second Stratification	15	12.6	53	53
Sum			117	89
			% reduction:	24%

entry concerning redundancies caused. There is no limit (other than total number of vertices) to the number of vertices whose incident arcs might be made redundant by entry of a given arc.

Because of the difficulty of devising algorithms for eliminating redundancies from a graph without exhaustive search for them, the problem is currently receiving a great deal of attention. Results to date indicate that the decomposition process described in a section below can help significantly in detecting redundancies, and that a few redundancies may be quite tolerable. If a small residue of redundancies can be tolerated, then there are fairly efficient processes for sweeping them out of the graph from time to time, and some simple additions to the search process could detect many redundancies as a byproduct of other manipulations. Furthermore, for graphs whose arc accession processes resemble random equiprobable arc generation, the region in which redundancies are probable is a small easily checked neighborhood, so that a rapid non-exhaustive checking process could eliminate most redundancies caused by entry of arcs.

3.5 Decomposition.

A significant portion of the research described by this report was concerned with the formulation and investigation of the decomposition principle. Decomposition will take on a new sense in this report in that it describes a technique for the representation of large graphs in the memory of a computer. It is essentially a technique for deriving successively less complex versions of a partiuclar graph from that graph, doing so in a way which makes the less complex versions useful as "shortcuts" in the search process. The expected effect on search effort is to make the effort required for any given search proportional to the logarithm of the separation between them, so that in large graphs, searches over long paths require relatively little effort.

The primary gain to be derived from this representation is a significant increase in search efficiency. It appears that it will be particularly useful for a graph which is searched frequently in response to queries concerning its content. The increase in search efficiency is bought at a cost of increased storage for the representation and the computations required to establish and maintain a decomposed graph. The balance between these advantageous and detrimental factors will ultimately determine the value of the technique.

3.5.1 Description.

3.5.1.1 Notation.

We continue the notation of the section on path number stratification with the following additions: The graph which represents the set of relations is denoted G_0 (V, A). It is assumed to be free of redundancies or, in other words, to be a basis graph.

The G_0 graph may be drawn as in Fig. 3-4. Here the vertices without precedents are at the left of the figure, vertices without subsequents are at the right. Every vertex is placed such that all its precedents are to its left and all its subsequents are to its right. In this section all graphs are drawn with this convention.

Usually the G_0 graph will be considered as a dynamic, growing graph. At any particular time it represents the current knowledge about a particular environment, all of the present statements involving some specific relation, as, for example, all known statements of the form "a is a subassembly of b". As time passes new information is learned about the environment and this is added to the graph in the form of new arcs. The dynamic nature of the graph complicates the problems of dealing with it and will be considered in this section as it affects the decomposition principle.

To put the reader at ease with regard to the very simple concept of decomposition an example is presented before a more rigorous explanation is provided. Consider the chain graph of Fig. 3-5a.

$$\begin{tikzcd}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0 \arrow[r] & 0
\end{tikzcd}$$

FIGURE 3-5a - CHAIN GRAPH.

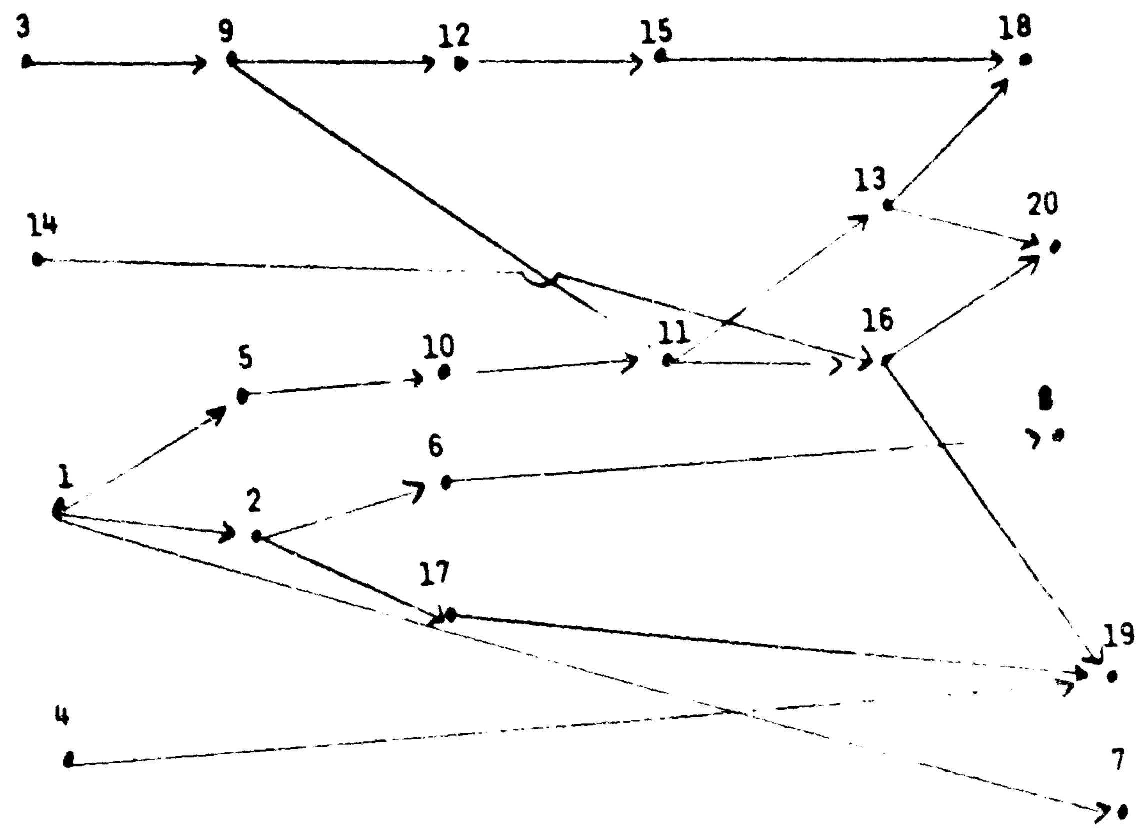


FIGURE 3-4 - THE G_0 GRAPH.

This graph is divided into four parts for decomposition as in Fig. 3-5b by cuts. The cuts are the dotted lines in Fig. 3-5b.

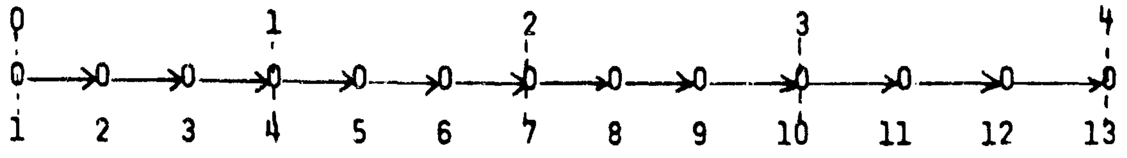


FIGURE 3-5b - CHAIN GRAPH DIVIDED BY CUTS.

Vertices v_1, v_4, v_7, v_{10} and v_{13} fall on these cuts and are called cut vertices. The set of cut vertices is called V_1. These vertices are used to form another graph, G_1. There is an arc in G_1 between vertices v_{ci} and v_{cj} if there is a path between v_{ci} and v_{cj} in G_n which does not include any other cut vertices. Thus, the G_1 graph for Fig. 3-5b appears in Fig. 3-5c.

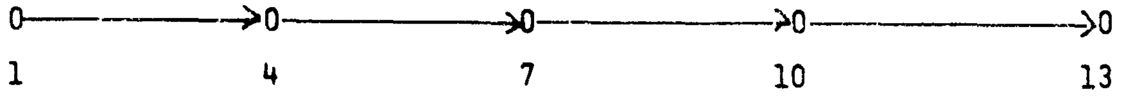


FIGURE 3-5c - THE G_1 GRAPH.

G_1 is called the first decomposition layer of G_0. Its existence can be very useful to a procedure which determines if there is a path between two arbitrary vertices. Say one asks whether v_2 is precedent to v_{12}. Of course for this simple example the answer is obvious but a systematic procedure must be provided to answer the question for a more complicated graph.

One technique of discovering the relation might be to trace a path forward from v_2 in G_0 and trace a path backward from v_{12} in G_0 until the two paths meet. This technique would require ten arcs to be traversed in G_0 in order to affect a meeting.

Using decomposition one can trace forward in G_0 from v_2 to cut 1 and trace backward from v_{12} to cut 3. The paths could then be traced in G_1 forward from v_4 to v_7 and backward from v_{10} to v_7. At v_7 a meeting has been found between the forward and backward paths. Thus, one can say that v_2 is indeed precedent to v_{12}. The important point here is that only 6 arcs were traversed to discover the relationship. If search time were proportional to the number of arcs searched, as it is assumed to be, the search time in the decomposed graph for this example is 0.6 that of the non-decomposed graph.

This is the savings of decomposition. Its costs are the extra storage required for G_1 and the calculations which are required to maintain the decomposition a graph growing with new information.

If the graph were large enough one could decompose the graph G_1 to a second layer with graph G_2. Decomposition might be applied a number of times on a very large graph.

3.5.1.2 Cuts.

For the chain graph of the last section a cut consists of a single vertex. When the vertex and the two arcs incident to it are removed from the graph two components of the graph are formed, one consisting of the vertices precedent to the cut vertex and the other consisting of the vertices subsequent to it. For more complicated graphs a similar phenomenon will occur but generally a cut will consist of more than one vertex, and a more rigorous definition is required.

Before this can be done two definitions are necessary. The precedent cut is the set of vertices which have no precedents. Thus for Fig. 3-4, v_1, v_3, v_4, and v_1 are said to be on the precedent cut. The precedent cut is called C_p. Similarly, the subsequent cut is the set of vertices which have no subsequents. For Fig. 3-4, v_7, v_8, v_{18}, v_{19} and v_{20} are on the subsequent cut. The subsequent cut is called C_s.

Now any cut can be defined as a set C of vertices which fulfill four conditions:

1) When the vertices in the set C and all the arcs incident to them are removed from the graph two sets of vertices are delineated X and W which have no arcs between them. Thus if $v_x \in X$ and $v_w \in W$ there is no arc (v_x, v_w) after the removal of the vertices in C and their incident arcs.

2) All members of the precedent cut will be members of X and the members of the subsequent cut will be members of W. Thus, if $v_x \varepsilon C_p$ then $v_x \varepsilon X$ and if $v_w \varepsilon C_s$ then $v_w \varepsilon W$.

3) The members of C are not related. Thus, if $v_a \varepsilon C$ and $v_b \varepsilon C$ then $v_a \nmid v_b$ and $v_b \nmid v_a$.

4) All the vertices in X are precedent to one or more members of C but are subsequent to no members of C. All the vertices in W are subsequent to one or more members of the cut but precedent to none.

Cuts using only the vertices of the graph will usually not sufficient for decomposition. The properties which define a cut are rather restrictive and most graphs will have fine sets of vertices which can fulfill them. In fact for Fig. 3-4 there is no set of vertices which can be a cut. To broaden the range of possible cuts, a new type of vertex is introduced called a <u>virtual</u> vertex. This name is to contrast with the vertices in the set V which will henceforth be called the <u>real</u> vertices of the graph.

A virtual vertex is placed on an arc between two real vertices as in Fig. 3-6. The addition of the vertex divides the arc into two arcs, one which enters the virtual vertex; one which leaves it. Thus, it has one immediate precedent and one immediate subsequent. The virtual vertex is identified by naming these vertices in the subscript of the vertex notation. Thus, the vertex inserted in the figure is called v_{a-b}.

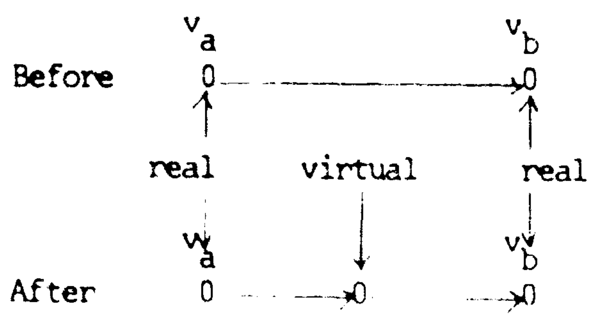
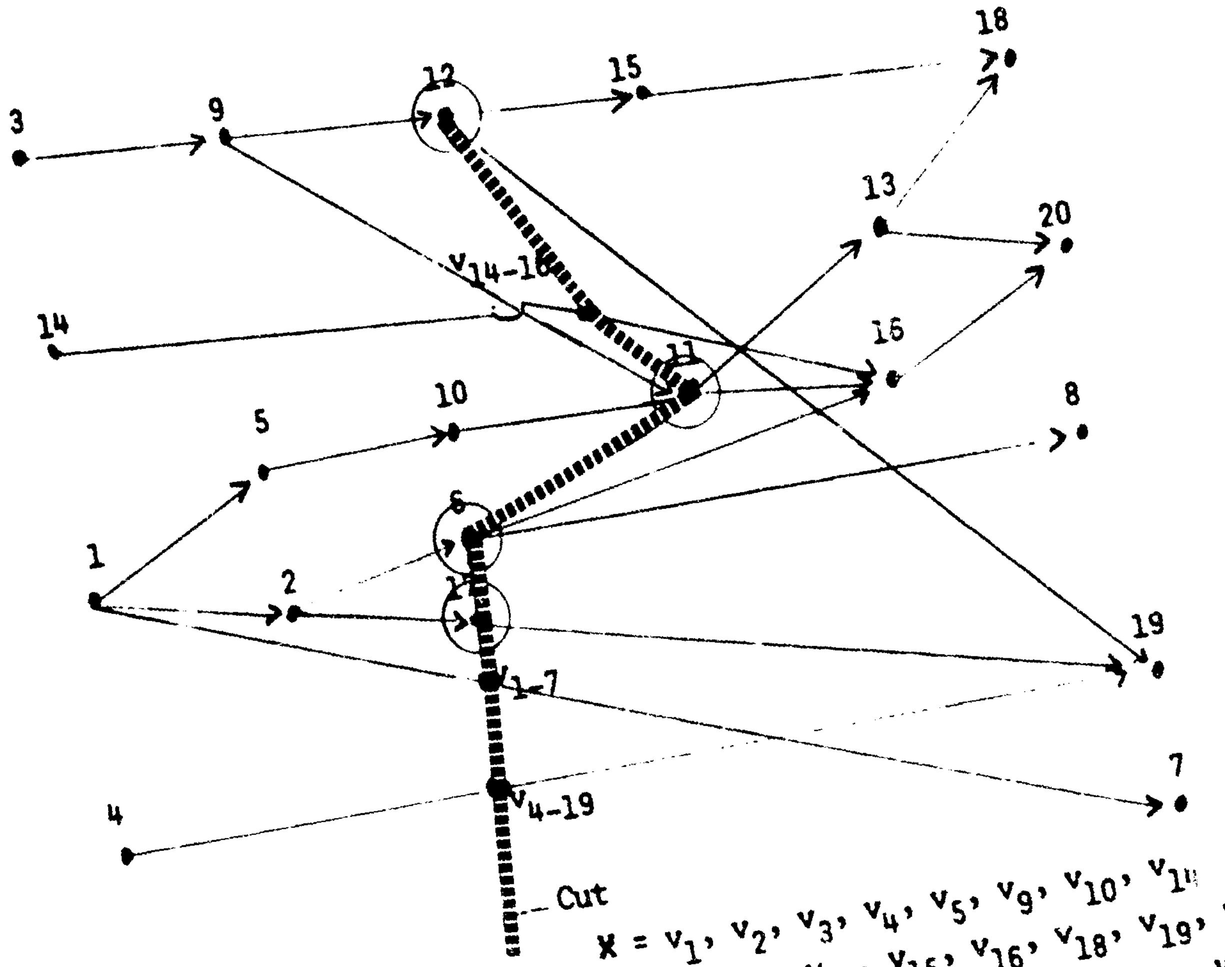


FIGURE 3-6 - <u>VIRTUAL VERTEX</u>.

Virtual vertices are only introduced to form a cut; hence they only appear in the set C, never in sets X or W. A cut including both real and virtual vertices appears in Fig. 3-7 indicated by a line through these vertices.



X = v_1, v_2, v_3, v_4, v_5, v_9, v_{10}, v_{1}

W = v_7, v_8, v_{13}, v_{15}, v_{16}, v_{18}, v_{19}, v_{20}

C = v_6, v_{11}, v_{12}, v_{17}, v_{1-7}, v_{4-19}, v_{14-16}

FIGURE 3-7 - AN EXAMPLE OF A CUT.

There are a number of disadvantages to including virtual cut vertices in a graph. From a search efficiency standpoint virtual cut vertices are undesirable, and they add materially to the memory space required for the G_0 graph and the decomposition layers. They are, however, unavoidable for our purpose.

3.5.1.3 The Use of Cuts.

Generally, G_0 will be a very large graph with many vertices and many arcs. Schematically such a graph might be represented by a rectangle as in Fig. 3-8 with the vertices and arcs not shown.

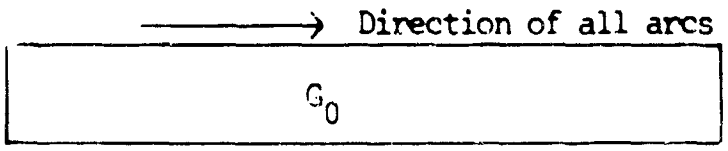


FIGURE 3-8 - SCHEMATIC REPRESENTATION OF G_0 GRAPH.

The direction of all the arcs will be assumed to be to the right as was suggested in Sect. 1. A cut will be used to divide this graph into subgraphs called G_0 subgraphs. The cuts will be indicated by vertical lines crossing the G_0 schematic as in Fig. 3-9. Cut vertices lie on these lines; since they are not related there are no arcs between vertices on the same cut.

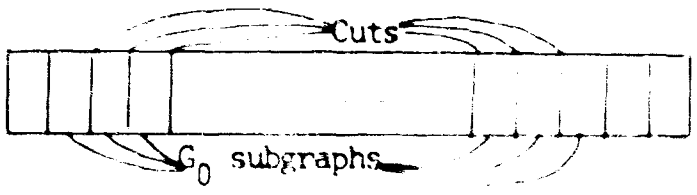


FIGURE 3-9 - CUTS IN G_0 GRAPH.

Two cuts border each subgraph. That which lies to the left of the subgraph is called the precedent cut of the subgraph and that which lies to the right, the subsequent cut. Vertices on the precedent cut can be only precedent to vertices in the subgraph and those on the subsequent cut can only be subsequent to the vertices in the subgraph.

Note for a large graph there are a number of cuts in G_0 and hence a number of subgraphs. When there is more than one cut, the requirement is made that all cuts must be disjoint, i.e., a vertex can be in not more than one cut in any layer. Cut vertices in G_0, both real and virtual, will form the set of vertices V_1 for a graph G_1. Paths which exist between vertices on different cuts in G_0 are shown as arcs in G_1. This graph could also be represented in schematic form as a rectangle as in Fig. 3-10.

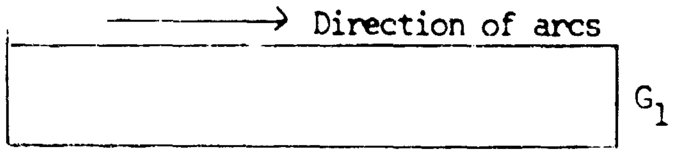


FIGURE 3-10 - THE SCHEMATIC REPRESENTATION OF G_1.

Generally, G_1 will have far fewer vertices than G_0 because, except for the virtual vertices, the cut vertices will only be a subset, usually a rather small subset, of the vertices in V. G_1 will also have fewer arcs than G_0 for every arc in G_1 represents one or more paths in G_0 consisting usually of a number of arcs. G_1 will also be assumed to be nonredundant. A consequence of this is that each arc in G_1 represents a path in G_0 which is incident to only two cut vertices and these form the initial and terminal vertices of the path.

Search will be considerably aided by the existence of the decomposed graph G_1. As an illustration consider the problem of searching for a relation between v_i and v_t in G_0. Then vertices are shown in the schematic of G_0 in Fig. 3-11a. The search operations are shown in Fig. 3-11b and 3-11c.

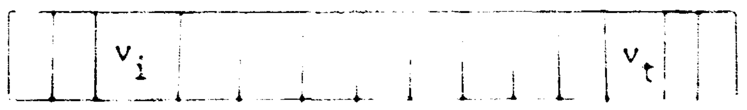


FIGURE 3-11a - VERTICES IN G_0.

Search might procede forward from v_i to the subsequent cut of the subgraph in which v_i resides. Concurrently one could search backwards from v_t to the precedent cut of its subgraph.

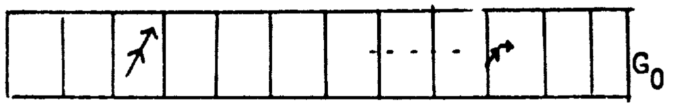


FIGURE 3-11b - SEARCH IN G_0.

Search could then enter the G_1 graph and quickly find the desired relation.

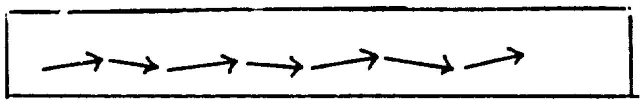


FIGURE 3-11c - SEARCH IN G_1.

Search is quickened because each arc traversed in G_1 would require the traversing of a number of arcs in G_1.

If G_1 forms a large graph, it too could be decomposed by the same technique into a graph G_2. It appears desirable that a cut in G_1 consist of the same vertices as a cut in G_0. Since the removal of any cut disconnects the vertices precedent to the members of the cut from the vertices subsequent to them, a cut in G_0 also fulfills the requirements for a cut in G_1; no new cut vertices are necessary.

The schematic of the G_0 graph with its cuts, the G_1 graph with its cuts and the G_2 graph appear in Fig. 3-12. The various graphs are called decomposition layers, G_1 is the first layer, G_2 the second, etc. A search in a decomposed graph to two layers is indicated in Fig. 3-12. Note that the search takes place in at most two subgraphs of any layer.

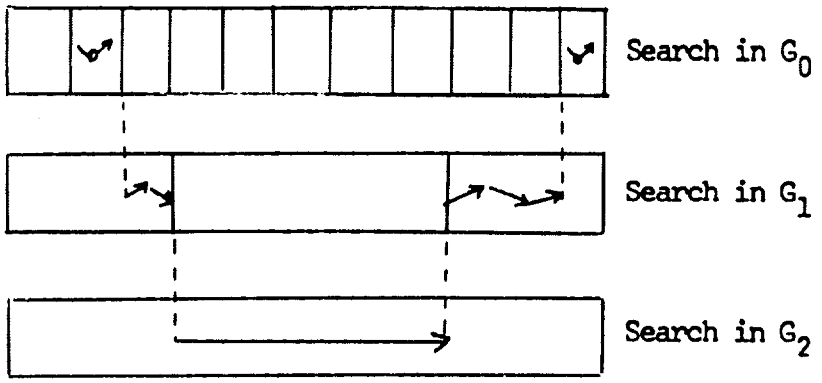


FIGURE 3-12 - SEARCH IN TWO DECOMPOSITION LAYERS.

The decomposition process can be applied to any layer, forming a G_3, G_4 and so on. This is the basis for calling it recursive, since G_2 is the decomposition of the decomposition of G_0, etc.

3.5.2 Tradeoffs of Decomposition.

The concept of decomposition seems to promise considerable gain in the efficiency of search in a large graph. The gains are not without cost, however, and one must not neglect the tradeoffs which arise between search efficiency, extra storage required, and the bookkeeping calculations needed to maintain the decomposition for a growing graph.

3.5.2.1 Storage Required.

The computer storage required for a decomposed graph includes that needed for: 1) the G_0 graph, 2) the virtual cut vertices which must be introduced, 3) the graphs of the decomposed layers introduced, 4) the identification of the membership of the cuts, and probably other less important factors which will be encountered during the coding of algorithms dealing with decomposed graphs. Because no computation experience is available for decomposed graphs, a discussion of quantities of storage required must be qualitative and somewhat speculative.

The G_0 graph appears to require the greatest bulk of storage. Of course, the greater part of this is required whether or not decomposition is used; hence, it is not a factor in judging the value of the scheme.

One aspect of the G_0 graph is changed with the incorporation of decomposition and that is the addition of virtual cut vertices. The presence of these vertices is undesirable from several points of view as will be later pointed out, so their number will be minimized. Most cuts of the G_0 graph will, however, include virtual cut vertices and they will cause an increase in the storage space required for G_0. The magnitude of the increase is dependent on the form of the G_0 graph and a good estimate will only be obtained with experimental experience.

Qualitatively, for decomposition to be a valuable technique for graph representation, the total number of cut vertices of G_0, both real and virtual, must be a small fraction of the total number of vertices in G_0. Both storage required and expected search time in the usual case are monotonically increasing functions of the number of vertices in a graph (graphs being considered comparable, for instance, if they have the same fullness ratio). The vertices of G_0 form the set of vertices for G_1. Thus, if the number of cut vertices in G_0 approaches the total number of vertices in G_0 before the cuts were made, the size of G_1 approaches that of the original G_0. A search in G_1 will now require almost the same time as a search between the same two vertices in the original G_0, obviating the advantage of introducing the decomposition. Only the disadvantages remain.

Thus, for decomposition to be of value, the total number of cut vertices must be a small fraction of the number of vertices in the original G_0 graph. Hence, if this condition holds, the number of virtual cut vertices must be an even smaller fraction and they will not add considerably to the storage required for G_0.

The preceding paragraphs touched on the storage problem for the first decomposition layer G_1. Since the storage required for a graph is roughly proportional to the number of its vertices and since in the practical case this number of G_1 must be a small fraction of that of G_0, the storage required for G_1 must be about the same fraction of the storage required for G_0. Say the $|V_1| = \frac{|V_0|}{n}$. When the G_1 graph is large enough it too will be decomposed into the second decomposition layer G_2. This is a similar process to the first decomposition; hence, one might expect $|V_2| = \frac{|V_1|}{n} = \frac{|V_0|}{n^2}$. If G_0 were very large; hence, decomposed a large number of times the total number of vertices required for all decomposition layers would approach an upper limit $|V_t|$ as the number of decomposition layers approach infinity.

$$|V_t| = |V_0|+|V_1|+|V_2|+ \ldots$$

$$= |V_0(1 + \frac{1}{n} + \frac{1}{n^2} + \ldots)$$

$$= \frac{|V_0|}{n-1}$$

There will always be a finite number of decomposition layers so, $|V_t| < \frac{|V_0|}{n-1}$.

If n = 11, which seems a reasonable goal for which tc strive, the total storage required for all decomposed layers would be less than 1.1 times the storage required for the G_0 graph. Thus, in all practical cases the extra storage space required for decomposition will be small.

3.5.2.2 The Maintenance of Decomposition.

If a graph were fixed and no changes were to take place, the cuts could be placed and the graph decomposed in an optimum manner, and there would be nothing further to do. But this study is dealing with the growing graph, constantly receiving new information about the environment which it is representing. This new information is given in terms of new arcs between vertices already in the graph and perhaps new vertices. Here the view is taken, for simplicity, that the vertices in the graph are fixed and only new arcs are added. (In fact, the introduction of new vertices into an existing list-structured graph representation poses no difficult new problems.) The new arcs create a variety of problems to a decomposed graph requiring constant maintenance so that the graph continues to fulfill the requirements of decomposition.

A new arc a_{it} added to a decomposed graph will fall into one or more of these five classifications:

1) The arc is redundant.

2) Both terminal vertices of the arc lie within the same G_0 subgraph and are not cut vertices.

3) One or both of the terminal vertices are cut vertices.

4) The terminal vertices fall into different G_0 subgraphs with $v_i < v_t$ before the arc was added.

5) The terminal vertices fall into different G_0 subgraphs with $v_t < v_i$ before the arc was added.

Class. 1 - Here it is assumed that if a_i is redundant it is not included in the graph; hence, no maintenance is required for this class.

Class. 2 - This class also requires no maintenance as no cuts are met or crossed by the new arc.

Class. 3 - This class may require some adjustments. To illustrate the situation consider the graph in Fig. 3-13.

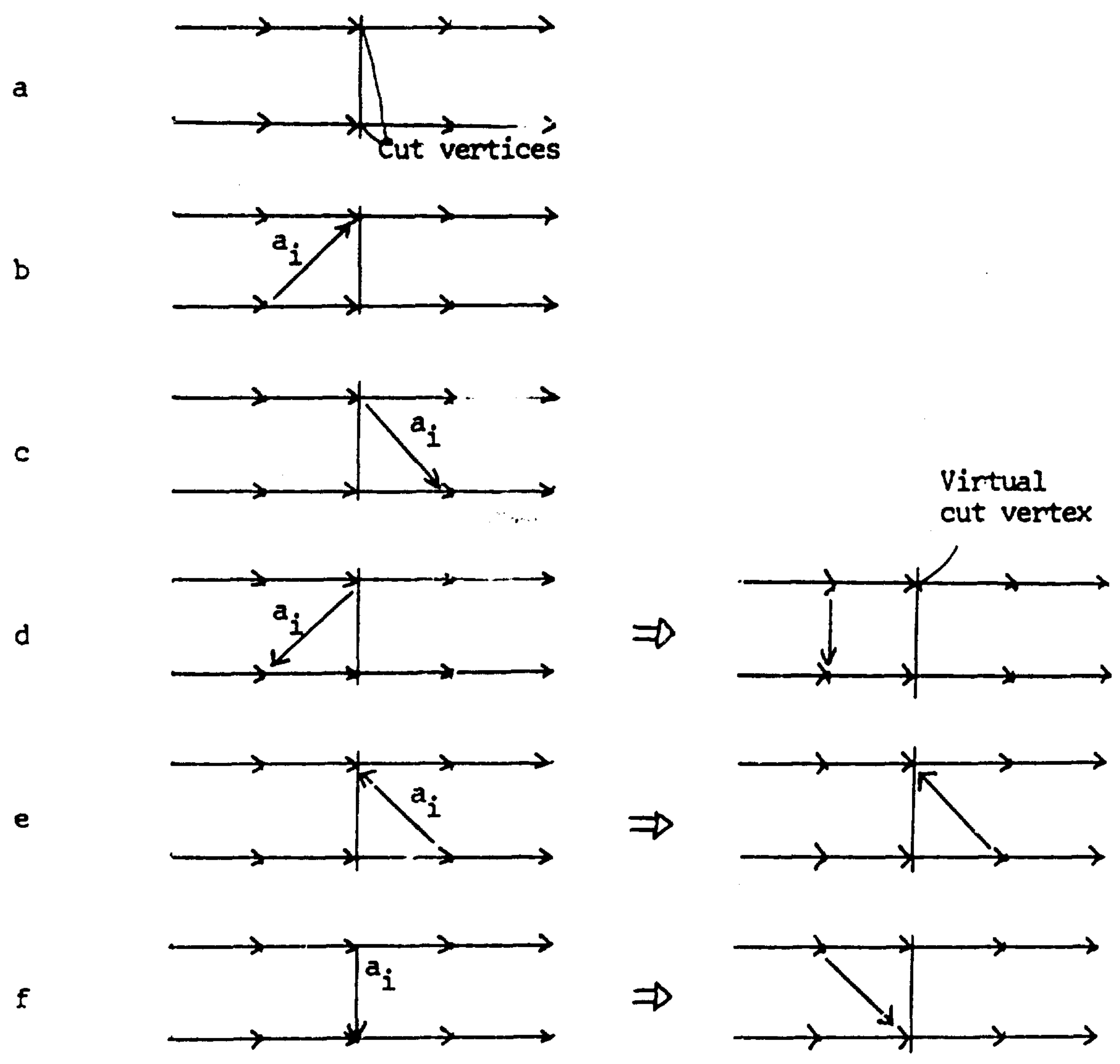


FIGURE 3-13 - THE VARIOUS WAYS AN ARC CAN BE ADDED SUCH THAT ONE OR BOTH OF ITS TERMINAL VERTICES ARE CUT VERTICES.

Figures 3-13b to 3-13f show four ways which an arc a_i might be added to this graph. For Fig. 3-13b and 3-13c, none of the restrictions concerning cuts are violated by the addition of the arc. The arcs added in Fig. 3-13d and

3-13e do cause one of the rules concerning real cut vertices to be violated requiring the modification noted to the right. Specifically, rule 4 of Sect. 3.5.1.2, above, is violated. The example of Fig. 3-13f violates rule 3 of Sect. 3.5.1.2.

If v_i and v_t are both real cut vertices but of different but adjacent cuts and $v_t < v_i$ before the addition of a_i, two virtual cut vertices would be added to correct the violation.

Class. 4 - This class and the necessary maintenance are illustrated in Fig. 3-14. In general, this type of addition will require the addition of a single virtual cut vertex for each cut crossed by the new arc.

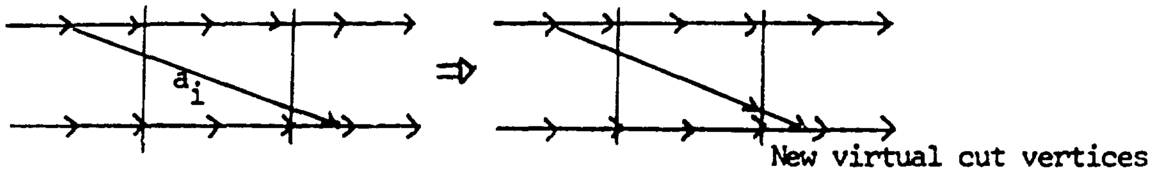


FIGURE 3-14 - THE MAINTENANCE REQUIRED WHEN A NEW ARC CROSSES A CUT.

Class. 5 - The addition of an arc falling into class 5 requires perhaps the most extensive maintenance. An arc in this class causes one or more vertices to violate rule 4 of Sect. 3.5.1.2. The modification indicated is to place v_t and all vertices subsequent to v_t and precedent to the subsequent cut of the G_0 subgraph of v_i into this subgraph. Fig. 3-15 illustrates this complicated maintenance rule.

It appears extemely difficult to implement this rule especially if the new arc passes over more than one cut in G_0. Evidence in the initial experiments has indicated that the occurrence of arcs traversing several cuts in the manner of Fig. 3-15 will be infrequent.

In a growing graph, when one uses decomposition, he must be prepared to expend computing time to maintain the graph, time which would not have been necessary had decomposition not been used. This then could be a significant factor in judging the value of the technique.

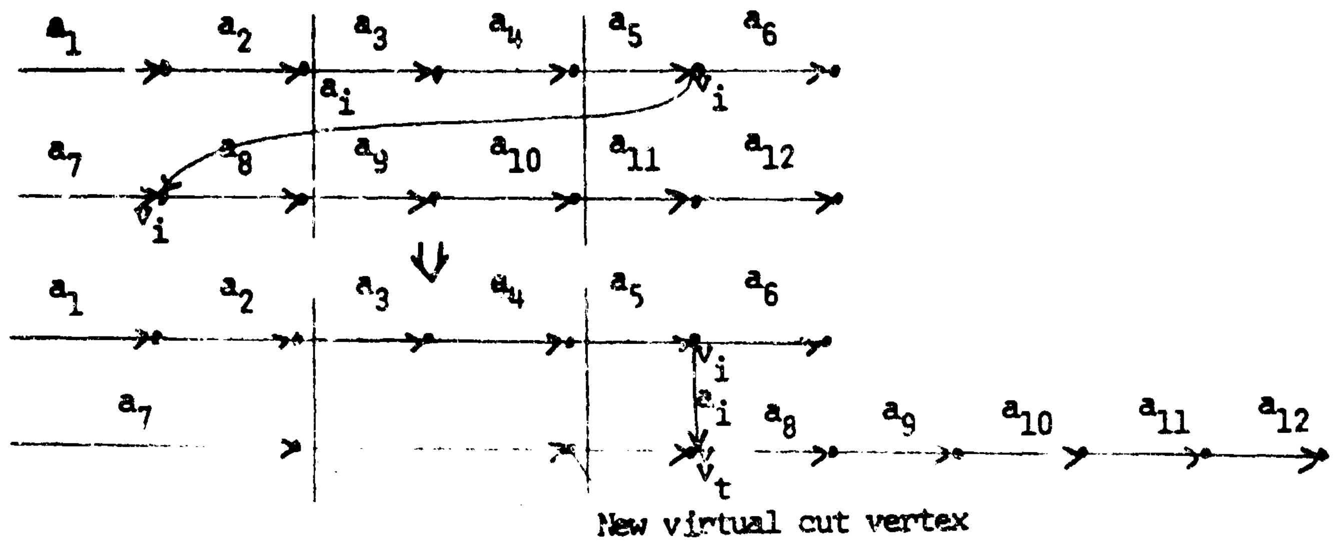


FIGURE 3-15 - M[illegible]TENANCE REQUIRED WHEN A NEW ARC CROSSES CUTS IN THE REVERSE DIRECTION.

3.5.2.3 Adding New Cuts.

In a growing graph the application of the decomposition technique is a dynamic process. Because arcs are constantly being added, if one started with an optimum placement of cuts, the optimality is soon lost without moving the old cuts and adding new cuts.

Constant upkeep to maintain optimality does not appear practical, as the optimization schemes presently envisioned are not trivial. One would not like to apply them after the addition of each arc.

An alternate, perhaps more practical approach, is to maintain all cuts once they are placed and only add new ones as the graph grows to a sufficient size. This implies a compromising of an optimal search decomposition for the sake of computational simplicity.

Section 3.5.3 will review further techniques and problems of forming cuts, but for the purposes of this section, it suffices to point out that the addition of new cuts is a computation introduced by the incorporation of decomposition and, hence, is a determinant of its value.

This section has considered several factors which are generally detriments to the value of the decomposition technique. They must be weighed against the advantages to search efficiency promised by this technique. A number of factors will affect the balance particular to the application to which the technique is to be applied. The investigators, however, feel that for many applications the advantages of decomposition will far outweigh its detrimental effects. Subsequent experimentation will test this opinion.

3.5.3 The Optimum Placement and Timing of Cuts.

3.5.3.1 The Optimum Placement of Cuts in a Fixed Graph.

In this section it is assumed that the complete graph G_0 exists and one would like to locate the cuts in G_0 to form the first layer with graph G_1. In like manner cuts are placed in G_1 to form the second layer with graph G_2, and recursively through higher layers of decomposition to form graphs G_3, G_4 ...G_j.... The criterion to be optimized by the cut placement is the expected search time in the decomposed graph.

To investigate the nature of this problem it is interesting to consider a simplified graph in the form of a chain. The vertices in the G_0 graph are specified by their longest path numbers S(i) which for a chain is simply a sequential numbering of the vertices.

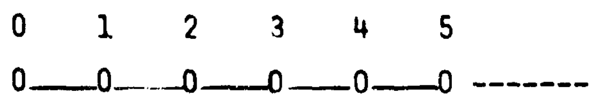


FIGURE 3-16 - CHAIN GRAPH.

The distance between two vertices v_a and v_b is defined as $|S(a) - S(b)|$. This is not the same as the search path length, for the search will take place in several layers. The search path length will be the total number of arcs traversed in the search. Some of these arcs will be in G_0, some in G_1, and so forth.

Decomposition of each of the layers will be accomplished according to the same rule. Thus, if there are five arcs between cuts in G_0 there will be five arcs between cuts in G_1. This is reasonable, for all the graphs of the decomposition are of the same nature and an optimal criterion for placing a cut in G_0 will also be optimal in the other graphs. Call X the number of arcs between adjacent cuts on any level.

The search algorithm between two vertices V_A and V_B ($V_A < V_B$) is affected in the same way as described in Section 3.5.1.3. Search will proceed forward from V_A and in reverse from V_B. Every time a cut is reached the search is raised to a higher level, until the extremes of the search in the forward and reverse direction are on the same level and between the same two adjacent cuts, or at a point where there is only one cut between them. Here a meeting between the searches will be affected if possible.

Of course the searchpath length between V_A and V_B will depend on the location of the two vertices in the G_0 graph. Considering the forward search path from V_A the number of arcs traversed from V_A to a cut in G_0 will be assumed randomly distributed with all possible search path lengths from 0 to X-1 equally likely. The mean number of arcs traversed in the forward search in G_0 will be $\frac{X-1}{2}$.. A similar mean will be true for the search from

V_B in the reverse direction, thus, the mean number of arcs traversed in G_0 during the search will be X-1. In like manner the number of arcs traversed in each of the higher layers less than layer J will be X-1. In layer J the forward and reverse searches meet. The number of arcs traversed in layer J may range from 1 to 2X. To ease the algebra the mean number of arcs searched in level J will also be assumed to be X-1.

For each change of layers we must pass from one graph to another. We assume that this requires a like effort to that of traversing an arc. Adding this factor, the effective mean number of arcs traversed in every layer is X + 1, again adjusting layer 5 by a constant for simplicity.

The distance of the shortest search which uses layer J is called $Y_0(H)$.

$$Y_0(H) = X^H \qquad (1)$$

$$\log_X Y_0 = H \qquad (2)$$

Except in the improbable case in which a search of distance $Y_0(H)$ actually uses layer H, all searches of this distance will use layers up through layer H-1. Thus, for searches over distance Y_0, $\left((\log_X Y_0) - 1\right)$ is a good estimation for the number of the highest layer used (including layer 0) and so $\log_X Y_0$ is a good estimation for the number of layers used.

We may assume on this suggestive basis that for searches of distance Y,

$$T = \log_X Y \qquad (3)$$

is the average number of lavers used over all possible searches of length Y.

From previous consideration, the total average search path lengths for searches of length Y is therefore

$$F = (X + 1)\, T \qquad (4)$$

Which may be rewritten

$$F = \log_e Y \left(\frac{X + 1}{\log_e X}\right) \qquad (5)$$

We see from this formulation that, for any cut spacing at all, the search effort turns out to be proportional to the logarithm of the separation between the vertices being examined rather than to the separation itself, an extremely significant saving for large graphs. Minimizing F analytically (in continuous variables) as a function of X only, we find that

1) The minimizing X is not a function of Y. In other words, a chain graph optimized for searches of any particular length is optimized for all search lengths.

2) The optimum X is the solution of $\log_e X = \frac{X+1}{X}$ or about $X = 3.6$ arcs between cuts. (In actuality, X must be an integer.)

A curve of F vs. X for an arbitary Y is shown in Fig. 3-17. We see that for the decomposed chain graph the optimum cut spacing does not depend strongly on cut spacing, so that, for example, a five-fold increase in spacing (from 4 to 20) only approximately doubles search effort. This suggests that a significant tradeoff between maintenance effort and search effort is possible and so is probably available in the more realistic cases as well. For the chain graph of Fig. 3-17, a search over a path having a length of 3,000 arcs in G_0 requires traversal of an average of about 29 arcs in an optimized graph and about 56 arcs in a graph with X = 20 arcs between cuts; these searches utilize 5.75 layers and 2.66 layers on the average respectively.

The results further suggest that below some limit the decomposition technique is not particularly helpful and should not be applied at all.

These results indicate that cuts should be surprisingly close together from the point of few path number differences. Most graphs, however, suffer from the fact that they are not chains and although the above result yield insights into the cut placement problem they cannot be practically applied. One factor which looms large in the effectiveness of cuts which cannot be accounted for in a chain graph is that search efficiency increases as the number of arcs in G_1 decreases for a given number of cuts in G_0. There is usually a large variation in the number of arcs introduced in G_1 by a cut of G_0 depending on the set of vertices included in the cut. The same comment holds for cuts made in higher layers of the graph.

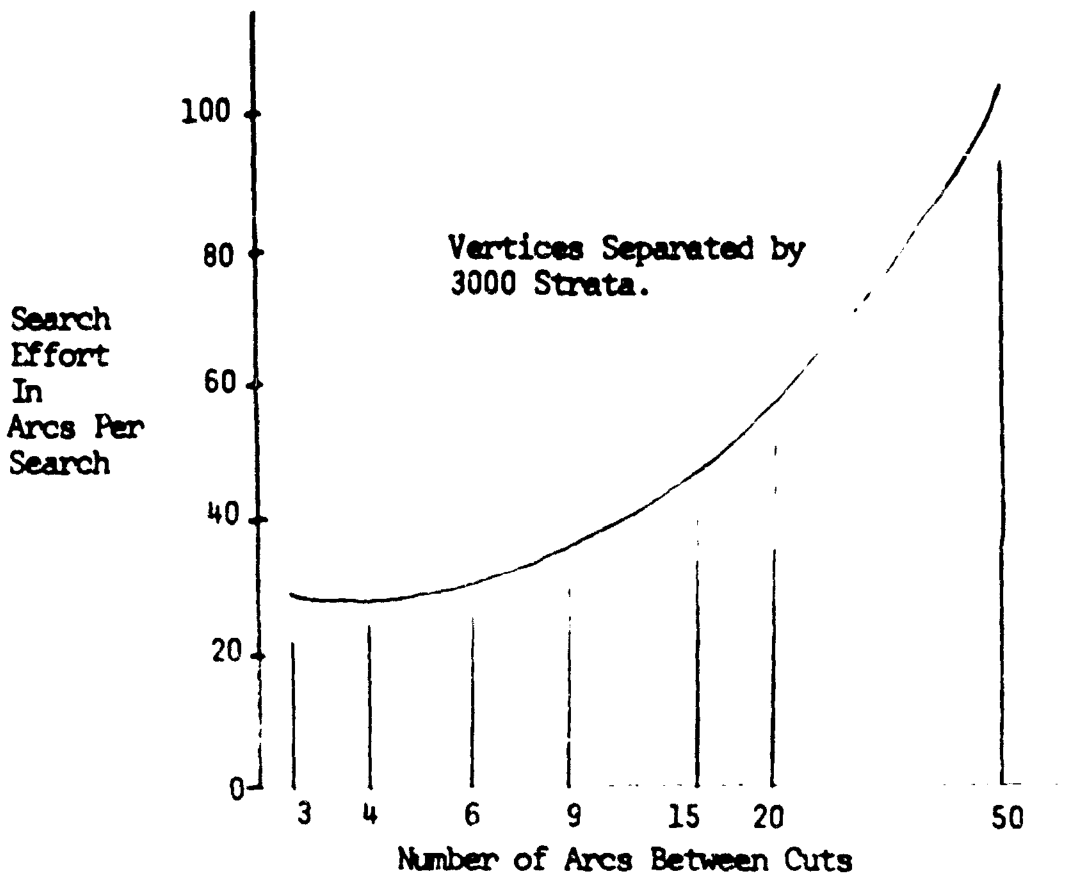


FIGURE 3-17 - SEARCH EFFORT IN A DECOMPOSED CHAIN GRAPH.

The problem of placing cuts in an optimal manner in a realistic graph is a difficult one. Research is currently almost completed for an algorithm which places a cut in G_0 between two existing cuts to minimize expected search time. The set of feasible cuts obeying the restrictions of Section 3.5.1.2 is too large for hand calculations for all but the simplest graphs but well within the power of a computer. The algorithm generates each of these feasible solutions and compares their effects on the expected search time. The feasible solution which yields the lowest expected search time is judged the optimum and used for the cut. A detailed description of this algorithm will appear in a future report.

3.5.3.2 The Dynamic Nature of the Problem.

In many cases the graph will not be fixed as suggested in the last section, but growing with arcs being added from time to time. This adds several new dimensions to the problem of placing cuts. The essence of the problem is that a particular arrangement of cuts which is optimum at one time will probably, after the addition of a few arcs, no longer be optimum. New arcs can affect the decomposed graph in the ways already described in Section 3.5.2.2. Sometimes these effects are marked. How to maintain the optimality of cut placement in the face of a growing graph is a problem which has not been solved.

The procedure currently envisioned is to keep all cuts once they are placed and to add new cuts as they are required between existing ones. This criterion for determining when new cuts are to be added is also at this time not presently available.

3.5.4 Experimentation by Simulation.

3.5.4.1 The Problem of Algorithm Evaluation.

To measure the value of the decomposition technique one must be able to measure search time, storage required, and the computational time required to establish and maintain the decomposed graph. Two main factors hinder these measurements. First, the time and storage required to implement these various aspects are very dependent on the algorithms used for the implementation. Section 3.3 indicated that there are several techniques for searching the decomposed graph. In like manner, there will be a number of ways to create and maintain cuts.

With some effort machine-independent factors, such as the number of arcs in a decomposed graph, can be separated from factors which depend on the particular experimental conditions. It is these machine-independent factors which are of greatest interest because they are meaningful in projections to other systems of programs and hardware.

A second factor in the value judgement is the dependency of the time and storage parameters on the particular graph which is the subject of the measurements. It appears that slight changes in the form of a graph can greatly change the efficiency of a particular algorithm applied to it. One can get around this problem by considering a universe of graphs and finding the expected values of the important parameters. The problem here is to define the universe of graphs of interest.

Applying classical mathematical techniques to the determination of the important expected values is generally fraught with difficulties. The problem here is the vast number of variables that one must specify to characterize a large graph. Perhaps one can sufficiently restrict the type of graph to be considered to provide mathematically tractable problems to sol But the results thus obtained leave one with doubts concerning their generality.

3.5.4.2 Simulation.

For this reason the primary means used for the evaluation of the decomposition principle and algorithms for dealing with it will be computer simulation. Programs have been developed which can generate graphs for study. Given a set of vertices, arcs are drawn sequentially and randomly from a set of feasible arcs. The size of graphs thus generated is presently limited by the size of the core memory of the computer. Our present experimental programs can generate graphs of about 400 vertices and 800 retained arcs.

The algorithms under study will be applied to the graphs thus generated. Facilities will be provided to count such important parameters such as path number occupancy, time for search and storage used. Of course, as other parameters become of interest these too will be calculated.

A number of graphs will be thus probabilistically generated and the algorithms applied to each. In this manner, frequency curves of the parameters will be plotted. From these one can measure the effectiveness of the various algorithms and ultimately the value of the decomposition principle.

3.5.5 Search Algorithms.

There are several interesting algorithms for searching a decomposed graph. The one which we plan to use for evaluation is described first below, and then others are considered in comparison.

3.5.5.1 Dual Recursive Alternating Search Algorithm.

Consider two separate search programs operating on a decomposed stratified graph in a list structured memory. For convenience, we will call one of them Jekyll and the other Hyde. Each of these is identified to the search routine described for stratified undecomposed graphs in an earlier section with certain exceptions:

1) Jekyll searches in the subsequent direction and Hyde searches in the precedent direction.

2) Each is able to detect vertices which have been reached by the other. This is a condition of the meeting of two parts of paths searched out separately, and so it is a detection of a relation between the terminal vertices of the search. It is a "termination condition" in addition to the others available, having the same qualitative effect as reaching the destination vertex in a search from an origin vertex.

3) Each is able to detect cut vertices. When a cut vertex is reached, if it is not a vertex reached by the other program, the program moves to the next higher layer at that cut vertex and marks a list associated with the entire set of vertices of that cut with a symbol indicating that it has reached that cut. If the other routine has reached that cut then the cut is marked as a "cut of meeting" and neither routine searches beyond that cut. Detection of "cut of meeting" becomes another terminating condition on both routines. Jekyll then calls Hyde; Hyde would call Jekyll.

The result in operation is that Jekyll searches recursively forward in G_0 until it reaches a cut; then Hyde searches recursively backward in G_0. If they do not meet or exhaust possibilities, Jekyll searches forward in G_1 and then Hyde searches backward in G_1, etc. The process is repeated until a relation is found or until possibilities of reaching the cut of meeting are exhausted.

As the reader may have guessed, Jekyll and Hyde are quite similar, and it is possible to implement them both by a single recursive routine with two entrances, a few parameters controlling the differences and a few separate lists for internal storage of the respective programs. It is an unusual recursive routine in that it effectively must compute its own name in order to enter the proper version of itself at each recursion or termination.

3.5.5.2 Other Possibilities.

Any other routine for search must allow for the fact that when a search proceeds into a higher layer, there is no a priori way of deciding when it should leave that layer for a lower one. For this reason all of the other alternatives here also search from both ends toward the middle. One way is to find all of the immediate subsequents of the origin vertex and put them on a list, and similarly for the destination vertex. The lists can then be compared and if no meeting is detected the process can be repeated for all of those vertices, again comparing lists and continuing (changing layers as appropriate) until a comparison of the strata reached indicates that all the paths from origin and destination have missed each other. Another termination condition is provided in that a search need not be continued beyond the stratum which contains the most remote active vertex for continuation from the other direction. By some controls in the algorithm, the difference between the active stratrum numbers can be minimized and the termination likelihood thus enhanced.

Alternatively, all paths in G_0 could be searched out to find all of the points of continuation on each side in G_1, and then the process applied to G_1 to the G_2 continuation, and so forth until meeting or until a common cut is reached from both search directions.

Either of these processes traverses all of the arcs in the required layers up to the cut of meeting which could possibly be on paths relating the two vertices. This seems to be a major drawback over the dual recursive approach since, in the latter, from each direction only one path is found at first, and others are found only if the first paths do not meet. Even then, other meetings are attempted in the highest layers before recourse is made to lower layers. The result would appear to be that in cases in which a relation was actually found the dual algorithm would tend to traverse far fewer arcs than the alternative routines, whereas for searches which did not detect a relation all of them would traverse about the same number of arcs.

4. EXPERIMENTAL RESULTS.

The purpose of this section is to report the results of Monte Carlo experiments which have been performed using some of the techniques described in previous sections, and to present the experimental method in enough detail so that the results can be interpreted. The experiments completed to date are not conclusive (in the sense that they allow reliable estimates of any of the statistics which characterize random graphs or particular ways to represent and manipulate them) but they are suggestive enough so that it was felt the report would be incomplete without some mention of them. The experiments deal only with graphs which have not been decomposed.

4.1 Graph Generation.

The mode of graph generation has an important bearing on the storage and search properties of the resulting graph and on the effects of any strategems for improving the properties. The mode chosen for experiment was intended to reflect the expectation that the graphs would be used for information storage, that there was an underlying finite, static "ultimate" graph whose structure was being revealed by stages, and that the accession of information was uncontrolled, or rather, controlled by circumstances other than the internal conditions of the graph. Thus, it was intended to reflect a class of problem-modeling situations and to be relevant to the storage and search needs involved in the use of computers in dealing with poorly structured problems.

The underlying structure selected was a total order of the vertices; an equivalent assumption states that information may be received for storage in the graph relating any two vertices whatever. The discipline of accession of arcs was such that all arcs consistent with the underlying structure were equally likely for accession at all times, independent of the graph content. The generation and search processes were programmed in LAP (List Assembly Program) language for the GE 225 or 235 computers. Each graph was first represented as an empty graph of N vertices and no arcs, each vertex having a list structure as exemplified in Table 4-1. For bookkeeping purposes, two auxiliary vertices were added to each empty graph, one precedent to all vertices and the other subsequent to all vertices, so that the resulting empty graph

TABLE 4-1

VERTEX LIST STRUCTURE

Address	Link	Symbol
Internal vertex name	a	Name of list of precedents
a	b	Name of list of subsequents
b	c	Stratum number
c	(As appropriate for a terminator of a list)	
Name of list of precedents	d	Name of precedent
d	e	Name of precedent
e	f	(as many as required, zero or more)
f	(As appropriate for a terminator of a list)	
Name of list of subsequents	g	Name of subsequent
g	h	(as many as required, zero or more)
	(As appropriate for a terminator of a list)	

Note: An empty list consists of one cell which is a term[illegible]ator.
Each line above represents one cell.

included 2N auxiliary arcs. This proved to be a significant simplification for programming, since contingencies of no precedent and no subsequent of a given (natural) vertex did not need to be provided for; as we shall see, the storage requirement is negligible. As information is added to such a graph, many of these arcs become redundant and so may be eliminated without altering the conditior that every natural vertex of the graph has a precedent and a subsequent.

A residue-class pseudo-random number generator was written for the 225 and was used to select arcs for entry into the graph. Equal intervals of the range of random numbers were assigned to each vertex. Pairs of random numbers were selected, and a corresponding arc was selected for entry provided that the two numbers were not in the interval for one vertex. The direction of inequality of the numbers was used to select the precedence relation of the arc, so that all arcs were consistent with an underlying total order of the vertices.* We define a parameter J, the number of arcs selected for entry up to any particul point in the graph growth.

Path number stratification was maintained continuously by a recursive routine which has already been described.

4.2 Storage Characteristics.

The characteristics of interest in describing the storage requirements of graphs include the following, all as functions of the number of vertices and the number of arcs selected for entry.

1) The probability distribution of the number of arcs in the basis graph.

2) The probability distribution of the fullness ratio of the graph.

* Some hand-generated random graphs were constructed using the order of appearance of the two vertices to control the precedence relation of each arc; an arc was rejected if it was inconsistent with the current state of the graph, transitivity and anticommutativity. Such graphs tend to assume a structure very close to a total order of the vertices very quickly in comparison to the graphs having a predetermined total order. This type of graph was not used in computer experiments because it seemed to be less representative of a problem-modeling discipline for accession of information, and because it was felt that techniques which would be effective for the type of graph which was used would be effective for this type as well.

3) The probability distribution of the maximum number of arcs in the basis graph.

4) The probability distribution of the number of entries which maximizes the number of arcs in the basis graph.

The data available to date are not adequate to support any meaningful statements about the forms, means or variances of these distributions. However, some specific cases can be presented.

Consider first the characteristic pattern of growth of a basis graph due to accession of randomly selected arcs. There are two limits which affect the relationship between the number of basis graph arcs A (G_B) and the selection count J. First, the basis graph can grow only as fast as the selection count; it tends to do so up to about J = N. Second, the final basis graph, representing a total order, has N-1 arcs, so the size of the basis graph must tend toward N-1 arcs. Figure 4-1 shows some cases of graph growth for various N. It indicates several tendencies which we expect to see supported by more extensive experiments. First, a tendency for the basis graph to reach maximum size around J/N = 3; second, a size limit of about 2N arcs; third, a tendency for the size limit to increase slowly with increasing N. At the point at which the underlying probability distribution of $\overline{A\ (G_B)}$ vs. J has a maximum, the qualitative effect is that on the average each selected arc which adds new information to the graph also makes one arc redundant, so that the basis graph size does not increase, and thereby average storage does not increase.

Reducing these indications to an estimate on possible use of 24K of core, each vertex has associated with it 10 cells allocated as follows: one cell for stratum number, two cells for reference to lists of precedents and subsequents, four cells (average) for reference to actual precedents and subsequents and three cells for the necessary list terminators. An estimated three cells per vertex of working storage for marking for fluctuation in total arc storage and for temporary lists during graph manipulation seems realistic. Thus, a graph of up to about 1,850 vertices (without decomposition) could be stored and used.

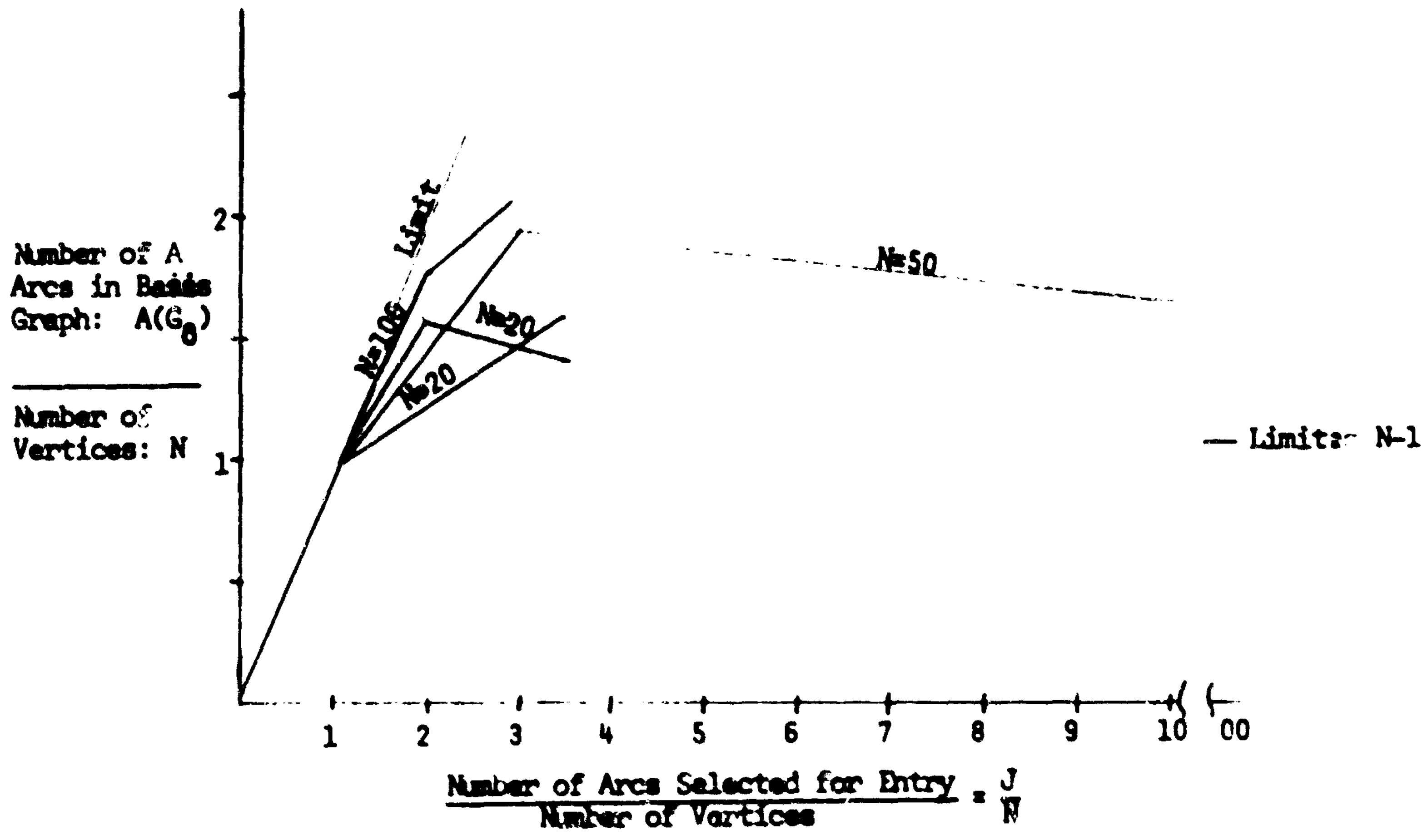


FIGURE 4-1 - GROWTH OF BASIS GRAPH.

Allowing one additional cell per vertex for naming the dominating cut, and about 10% of the space for all of the decompositions of G_0, a decomposed graph might have up to 1,550 vertices in 24K words of core. All of the above assumes one list cell per word of memory, which is certainly feasible on the longer wordlength machines.

As described above, it is a decided programming convenience to let every vertex of the graph being represented have at least one precedent and one subsequent. For this purpose, two auxiliary vertices are created and set precedent and subsequent to all N vertices during the creation of the empty graph, at J = 0. Arcs from these vertices rapidly become redundant (considering the structure of the N + 2 vertex graph) as the graph grows, and so may be eliminated. Figures 4-2 and 4-3 show a specific example of growth of an experimental graph in which both the basis graph and total storage were tabulated. Figure 4-2 shows specifically how the basis arcs tended to replace auxiliary arcs so that the overall storage requirement was nearly constant and was decreasing toward the total-order limit at the end of the experiment. Figure 4-3 shows the growth of the corresponding fully explicit graph and the significant saving in required storage. Since the number of arcs in the basis graph approaches N-1 while the corresponding number in the fully explicit graph approaches $\frac{N(N-1)}{2}$, the storage advantage of representing only the basis graph (taken as the limit of $A(G_E)/A(G_B)$ increases linearly with N. The figure is based on estimates of the fullness ratio using the number of basis graph arcs retained, as described in the section on estimating fullness ratios. The list structures compared were similar, but storage for path number stratification was not included in the fully explicit graph. This accounts for the difference in sizes of the two empty graphs at J = 0.

The experiments to date have not produced any reliable predictor for fullness ratio as a function of J. Several probabalistic models of graph growth have been constructed by the usual procedure of making unjustified

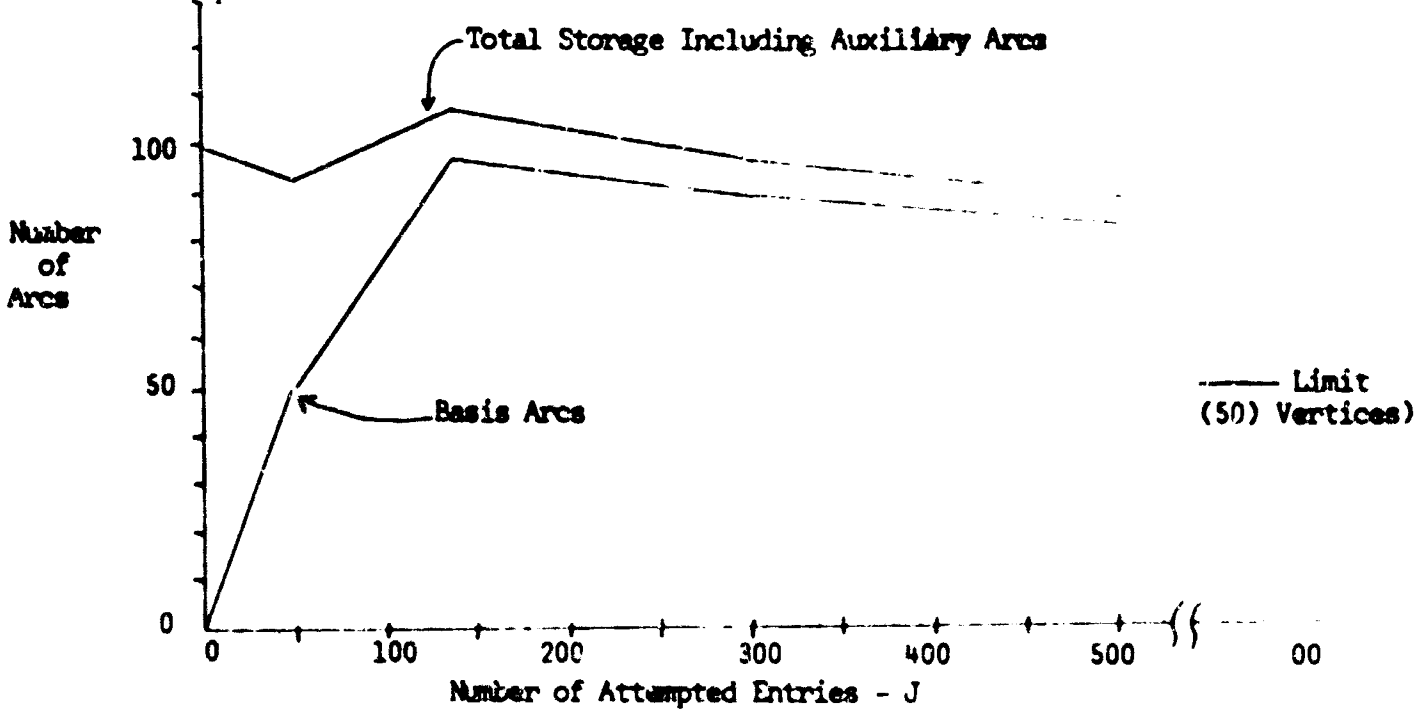


FIGURE 4-2 - EXAMPLE OF STORAGE REQUIREMENTS DURING GRAPH GROWTH.

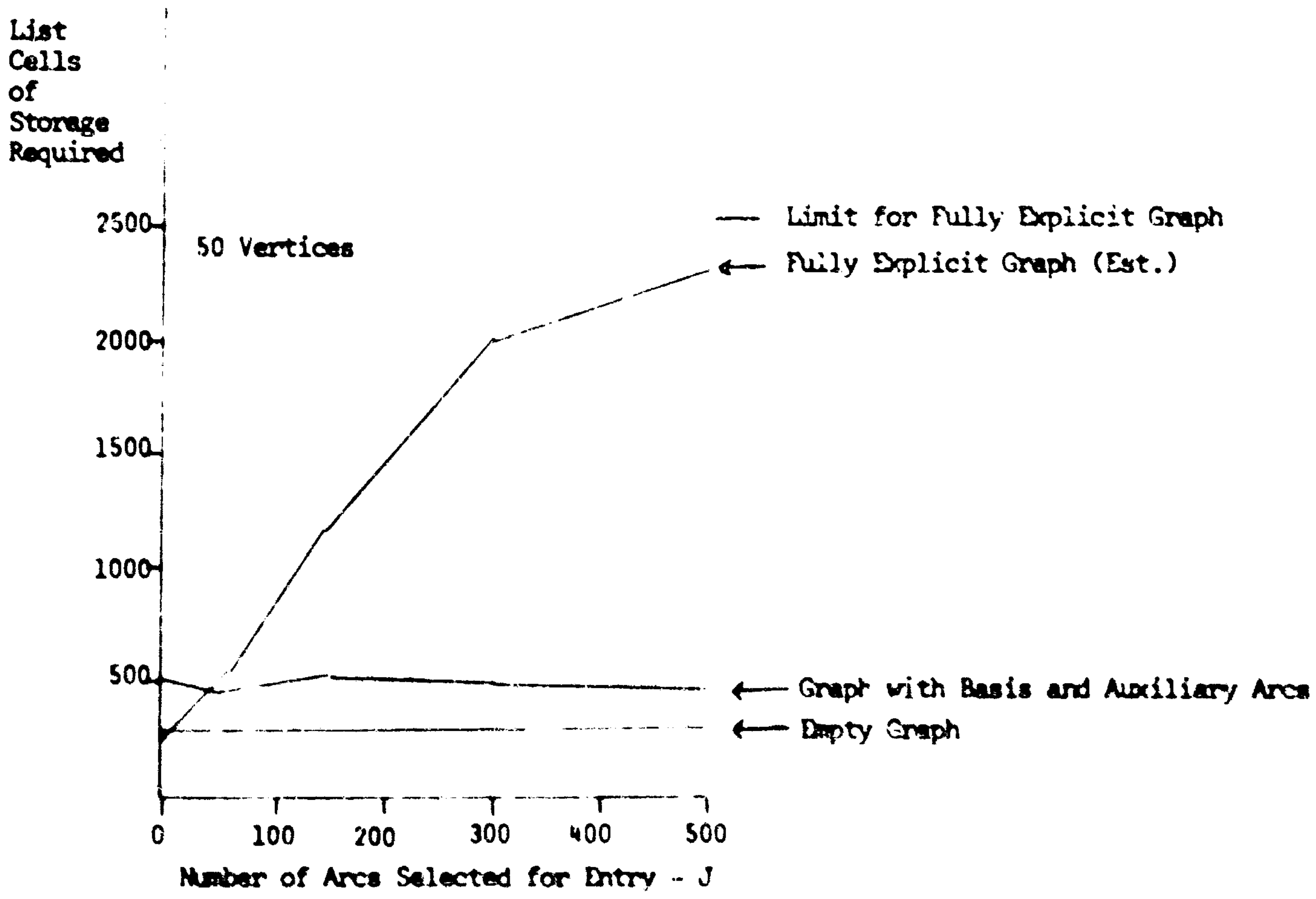


FIGURE 4-3 - EXAMPLE OF STORAGE REQUIREMENTS DURING GRAPH GROWTH.

(and hopefully, unimportant) assumptions of independence; none has proved to be realistic. The use of better estimators of F(G) in current experiments may lead to a useful predictor.

4.3 Search Characteristics.

The algorithm for search in a fully explicit graph is comparable to the recursive algorithm with the added termination condition: "Always terminate instead of any recursive continuation." This is effective because any vertex which is related to a given vertex is connected to it by some single arc in the fully explicit graph. The effort of a search to confirm or deny a given randomly selected relation in the graph may be estimated as follows. For a randomly selected vertex, the number of sequents in a chosen direction is

$$\underbrace{F(G)}_{\text{fullness ratio}} \quad \underbrace{(N-1)}_{\substack{\text{number of other}\\ \text{vertices}}} \quad \underbrace{(.5)}_{\substack{\text{for the choice of one of}\\ \text{the two sequent directions}}} \quad \underbrace{(Y)}_{\substack{\text{success}\\ \text{factor}}}$$

where Y = 1 for searches which fail to find a relation and Y = .5 for searches which find a relation. We may use this formula to estimate the search effort for any graph with known fullness ratio.

None of the experiments to date has included collection of comparable statistics on a fixed graph for the recursive search process used. However, some statistics are available for intervals of graph building. Two are tabulated as examples in Table 4-2; fullness ratios used for each interval were the mean of the estimated ratios at the interval limits. The case chosen is an unfavorable one for the stratified basis graphs, since it does not take advantage of the fact that over half of all the possible randomly selected searches will be answered on the basis of stratum numbers alone, nor that, for searches for any relation, the fully explicit graph must be searched in two directions. In spite of these unfavorable factors the experimental searches took comparable or less effort than the estimated searches in the fully explict graph.

The indication that the search effort has not been materially increased is encouraging. It remains to verify the results over a significant range of graph sizes and fullness ratios.

TABLE 4-2

EXAMPLES

	Fully Explicit Graph			Stratified Basis Graph		
	Mean Fullness Ratio	N	Estimated Effort in C_E	Number of Experimental Searches	Number of Arcs Traversed	Average Actual Effort
Successful Searches	.436	20	2.07	5	11	2.2
	.479	20	2.27	3	7	2.30
	.52	50	6.37	121	300	2.5
Unsuccessful Searches	.436	20	4.14	19	55	2.9
	.479	20	4.55	16	61	3.84
	.52	50	12.7	277	1619	6

One of the factors which affects the search process significantly is the length of the paths of search. The longest path involved in a given search determines the maximum depth of recursion, although it does not determine the number of times the routine will recurse. A measure of path length is afforded by the maximum path length in the graph, which, of course, is easily determined from the stratum numbers. Figures 4-4 and 4-5 show some examples of the changes in path length and path number occupancy respectively. Similarly, the occupancy levels of the various strata affect the numbers of paths which take on various lengths.

4.4 Determination of the Fullness Ratio of a Graph.

The purpose of this section is to present several techniques for determining the fullness ratio of a graph. To review, the fullness ratio of a graph G having N vertices is defined to be

$$F(G) = 2 \frac{A\ (G_E)}{N\ (N-1)}\ ,$$

the ratio between the number of arcs in the fully explicit graph G_E which corresponds to that graph and the number of possible arcs N (N-1)/2. It is the proportion of all of the relations represented by a total ordering of N vertices which is represented in the given graph. It may be determined in several ways, including:

1) direct determination,
2) independent estimate,
3) various "byproduct" estimates.

Direct determination of the fullness ratio would be an exhaustive process whereby every pair of vertices in the graph would be tested for relation and the calculation made. It involves some process equivalent to constructing the fully explicit graph and counting its arcs. For large graphs the process involves a great deal of computation; for graphs using most of a list-organized core memory, memory may be exceeded.

A possible estimating procedure is the following:

1) A number of pairs W of two different vertices are selected randomly.

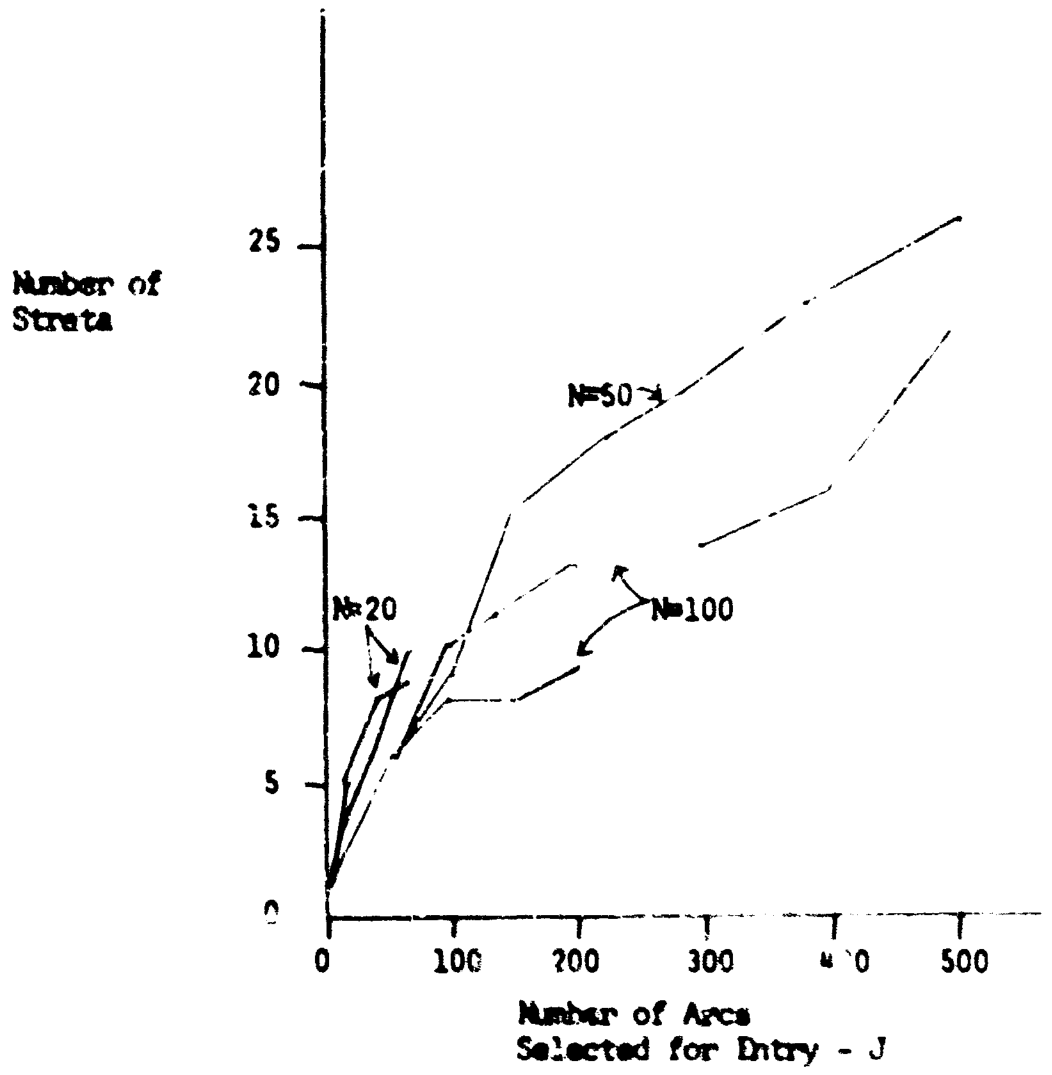


FIGURE 4-4 - CHANGES IN STRATIFICATION DURING GRAPH GROWTH.

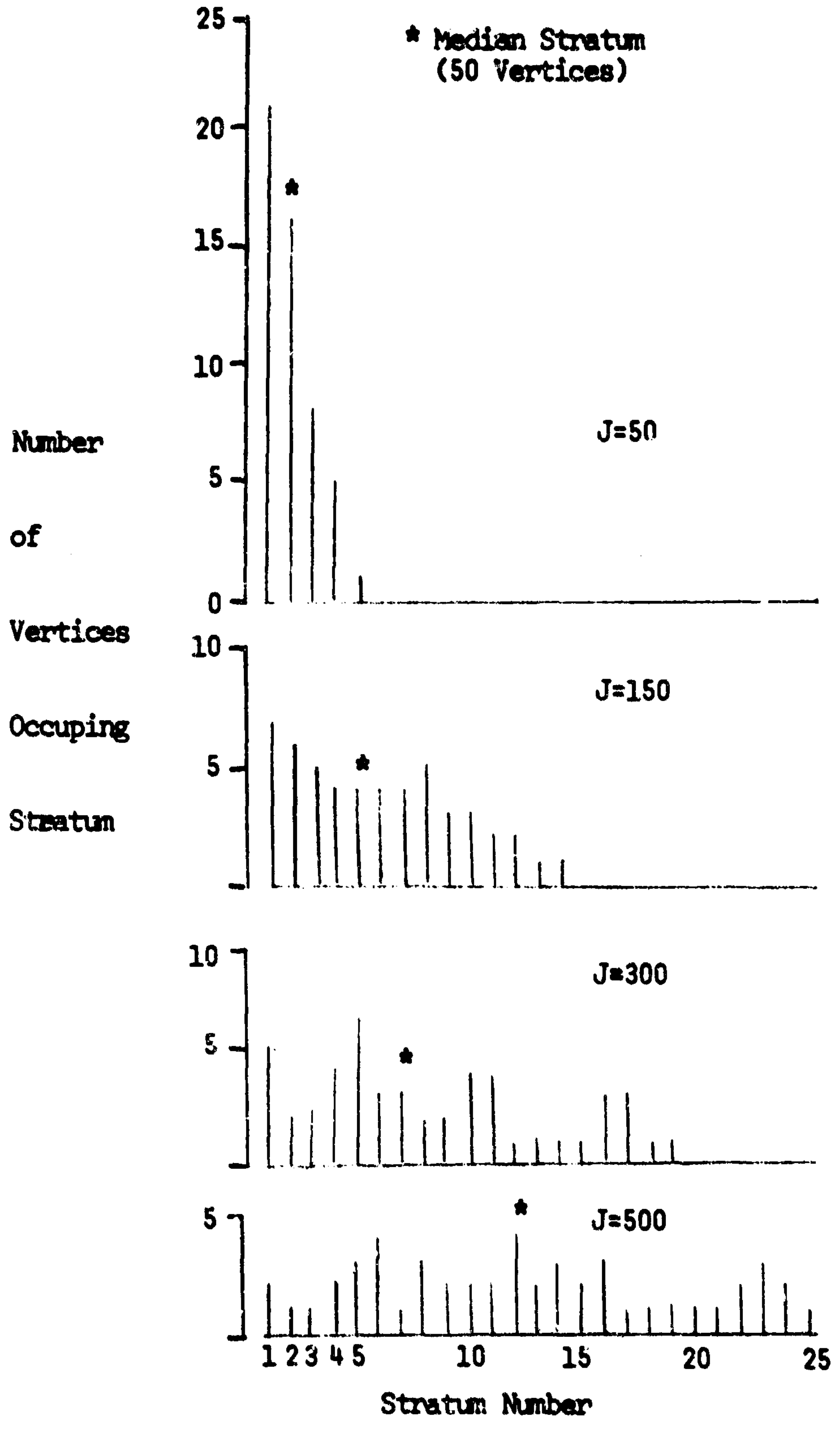


FIGURE 4-5 - EXAMPLE OF CHANGE OF STRATUM OCCUPANCY DURING GRAPH GROWTH.

2) Graph G is examined to determine the number of pairs X which are related in it.

3) Compute $F^*(G) = \frac{X}{W}$

The estimating procedure is justified as a direct sampling in the set of all possible arcs. For every pair W_i there is one relation which, although it may be unknown, is either represented in G or not, and its representation is determined from G. The quantity is binominally distributed with

mean $\overline{F^*(G)} = F(G)$, and with

variance $\frac{F(G)\ (1-F(G))}{W}$

The process of generating a random graph produces several opportunities for estimating the fullness ratio at various stages of completion. Two of these types of estimates are described below. In the first, some small fraction of the total number of arcs which were generated for entry, as for example the last 10%, may be taken in place of set W in the previous procedure. The proportion rejected as being redundant is taken to be the estimator of the fullness ratio. An assumption behind this procedure is that the fullness ratio did not change significantly during the entry interval selected, whereas, in fact, there is no assurance that this is the case.

A second possibility is to consider the set of all arcs selected for entry, and to use the proportion of that set which is not included in the resulting basis graph as an estimator of the fullness ratio, on the ground that it represents the redundant fraction of a randomly selected set of arcs. This estimator is surely subject to strong dependence on the graph produced. It is likely to have a very high variance as well; it can be shown that the estimator will consistently be above the mean fullness ratio (for a given number of selections) whenever the actual ratio for the particular graph is below the mean and vice versa. The preliminary results given below for $F^*(G)$ are based on these two byproduct estimation methods, since neither exhaustive determination nor an independent estimating procedure were included in the tests used.

A third "byproduct" estimating procedure is rapid and appears to be relatively accurate for $F(G) > 0.25$: If the longest path contains M arcs, and if we assume that every vertex is on some one longest path, then each vertex is related to M others and there are $\frac{MN}{2}$ arcs represented in the graph. Then

$$F^*(G) = \frac{MN}{2} \cdot \frac{2}{N(N-1)} = \frac{M}{N-1}$$

Since this estimator apparently tends to give a slight over estimate, the estimator

$$F^*(G) = \frac{M}{N}$$

is preferable because it is simpler and apparently somewhat more accurate.

For statistically meaningful tests the independent estimate appears to be the most attractive in terms of effort and validity.

In addition, a lower bound on the fullness ratio can be determined from a graph's stratification, since the M arcs on the longest path form a subgraph whose fully explicit graph consists of

$$F^B(G) = \frac{M(M + 1)}{2}$$

arcs. Since these arcs are surely represented, $F^B(G)$ is a lower bound on $F(G)$.

4.5 A Heuristic in Discovering Problem Structure.

During the experiments we noticed that there were apparently no instances in which a random graph developed two independent components (subsets of vertices not connected by any arc) which were both of significant size (more than about 3 vertices) and which together included most of the graph. The overwhelming tendency is for one major component to be formed, and for that component to take up unconnected vertices and components consisting of one or two arcs without any major second component ever being formed. The formation of this one component which covers nearly all of the graph occurs after only a small amount of information has been entered, well under N arcs in all the cases studied.

The probability distribution of the number of components of graphs constructed by the selection technique used in our experiments has been previously published.* This reference contains an approximating formula for the expected number of components which have j vertices for a graph with N vertices after N selections of random equiprobable arcs have been made. It is:

$$E\,(S_j) = \frac{2^{j-1}\, j^{j-3}}{(j-1)!}\, Ne^{-2j}.$$

In the course of deriving the approximating function it is shown that for the smallest graphs, only the one large component and the isolated vertices contribute significantly to the mean number of components, that for somewhat larger graphs only these and the isolated pairs are significant, that for yet somewhat larger graphs the isolated triples become significant, etc. The muliple-large-component case never contributes significantly.

On the basis of the above, we may pose the following heuristic: Whenever a growing set of relations develops into a graph with two major components, the arriving relations are not equiprobable and there is some underlying structural difference between the two sets of vertices and some kind of restriction on the arrival of relations between the two sets. Certainly under some conditions this is generalizable to more than two sets.

This is a particularly interesting heuristic because it imparts information about underlying structure on the basis of a very small amount of information compared to the information necessary to define the underlying structure.

* See reference 3. The paper contains expressions for the exact distribution and other approximating functions as well. It would be interesting and relevant to extend these results to develop statistical decision procedures on hypotheses concerning underlying structures and arc entry probabilities based on the component configurations of the resulting graphs.

5. CONCLUSIONS AND PROSPECTS.

The study reported here has developed an approach to representation of directed graphs which appears to provide significant economies of computer processing effort and storage space over previously documented techniques for large graphs. In the directed graphs studied, each arc represents a relationship between the two information entities at its terminals; the relationship can be any transitive, anticommutative relationship such as "is greater than" or "must precede". The key features of the representation scheme are stratification of a graph based on the lengths of paths in the graph, a recursive decomposition technique which produces successively less complex representations of a graph, and a recursive search technique which utilizes the stratification and decomposition. Preparations and preliminary results of tests of the statistical properties of the representation based on such graphs with randomly chosen arcs are encouraging but not yet definitive.

The representation approach is intended to be useful in a computer-use context in which there is an on-line conversational process dealing with problem definition as well as problem solution. The user is given the capability to define relations between information entities and to accumulate instances of these relations in individual data bases available to him and to the processes he uses. These data bases provide among other things, a repository of qualitative information which can be used as a basis for decisions in a jointly-developed problem model.* The relational data bases are, in a sense, a complement to the attribute-value descriptive capabilities of list processing languages.

* The representation may also have some value independent of man-computer processes. For example, large job-shop precedence requirements, time precedence graphs for partly-parallel computations, dependence relations between subroutines under an executive system, partly defined priority structures and relations between natural-language semantic categories might be usefully represented using these methods without manual intervention.

In such a context the user could declare relations just as variables are presently declared to be real, Boolean, string, etc., in algorithmic languages, and symbols assigned for use in communication concerning information entities which may be so related. His processing facilities would allow him to call for and insert information in his data base in terms of the relations he declares. He could also write logical expressions whose values depend on stored relationships just as he can now write logical expressions in algorithmic languages whose values depend on stored numbers and their equalities and inequalities.

What set of relations should be provided? A simple, easy-to-learn set which is complete enough so that one may use it for anything. We have attempted to enumerate the logically different relations which could be defined using only three symbols (the minimum required for defining such important properties as transitivity). The enumeration was not completed but it was carried far enough to show that the number of relations which could be so defined is probably at least several hundred, taking into account both acceptability of combinations of relations and inferences from combinations concerning other relations on the same symbols. It is unreasonable to provide several hundred mathematical forms for man-machine communication because massive confusion on the part of the men would result.

One guide to the forms of the most useful relations is the set of relations which are represented by simple relational phrases or sentences in natural languages. We may examine English, for example, and find relational phrases such as "exceeds", "is a parent of", "equals", "is near", each of which has a different underlying logical form of allowable combinations and valid inferences. It seems particularly important to provide forms which correspond to the frequently used relational phrases. Among these would be the transitive, anticommutative relations (the subject of this report) such as "precedes", "is more important than", "is a subset of", etc., the transitive commutative relations such as "equals", "is with", the non-transitive, non-commutative relations such as "differs from" and other relations used to describe a linear field (such as positions of a set of objects in three-dimensional space.) An examination of the logical forms of relations which have been discovered by natural language analysis may disclose other frequently used forms.

Such a basis is unsatisfying in that it does not provide any assurance of completeness for problem modeling purposes. A satisfactory compromise would be to discover a basic set from natural language analysis and then to augment these with a minimum set of additional relations such that every enumerable relation on three symbols could be defined as some use of at most two of the basic relations. The size of such a minimum set is unknown; hopefully it is small. At any rate, the forms of some of the relations which must be provided are obvious.

As a long range prospect, those relations which prove particularly useful can be implemented in hardware as particular kinds of associative memory, with the associational property and internal inference taking the logical form of the corresponding relation in each case.

The specific effects of representing directed graphs as discussed in this report should be documented in a convenient form for prediction and comparison in future experimentation by others. For this purpose the computer porgrams used for the present preliminary results will be extended to provide more extensive statistical information on stratified graphs and to allow decomposition, including experiments on when and where to cut during the decomposition process. The programs are also to be modified to produce specific information on the effort required by the decomposition process and subsequent maintenance during graph growth.

It would be possible to extend the experiments to graphs having an underlying form other than total order, or having unequally probable accession of different arcs. The variety of such possible extensions is endless and their utility is questionable at best. Furthermore, many of the possibilities can be estimated as compositions of the kind of graph already studied. Therefore, we have no plans to consider other underlying structures or generation probability disciplines for transitive, anticommutative relations.

In summary, the results to date appear to be a useful step toward a goal of making available methods of easy man-computer communication concerning relations between information entities. After completing study of the effects of stratification and decomposition, another relation will be selected for representation.

REFERENCES

1. Raphael, Bertram; "Semantic Information Retrieval", Thesis at Massachusetts Institute of Technology, Jun 1964, AD 608 499.

2. Bobrow, D. G. and Raphael, B.; "A Comparison of List-Processing Computer Languages", Communications of the ACM, Vol. 7, No. 4, pp 231-240.

3. Austin, T. L., Fagen, R. E., Penney, W. F. and Riordan, J.; "The Number of Components in Random Linear Graphs", Annals of Mathematical Statistics, No. 30, 1959.

The references below concern related work whose purposes or methodology are closely related to those of this project.

4. Simmons, R. F.; "Storage and Retrieval of Aspects of Meaning in Directed Graph Structures", System Development Corporation, SP-1975/001/02, September 1965, AD 622 017.

5. Levine, Roger and Maron, M. E.; "Relational Data File: A Tool for Mechanized Inference Execution and Data Retrieval", The Rand Corporation, Memorandum RM 4793PR, December 1965, AD 625 409

LIST OF TABLES AND FIGURES

LIST OF TABLES AND FIGURES (contd)

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Command and Control Company Suite 1315, 1750 Pennsylvania Avenue, N. W. Washington, D. C. 20006	2a. REPORT SECURITY CLASSIFICATION Unclassified	
	2b. GROUP --	
3. REPORT TITLE A Data Structure for Directed Graphs in Man-Machine Processing		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Annual Report		
5. AUTHOR(S) (Last name, first name, initial) Mann, William C.; Jensen, Paul A.		
6. REPORT DATE June [illegible], 1966	7a. TOTAL NO. OF PAGES 66	7b. NO. OF REFS 5
8a. CONTRACT OR GRANT NO. NOnr 4815(00) b. PROJECT NO. RR 003-10-02 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 77-106-1 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None	
10. AVAILABILITY/LIMITATION NOTICES Distribution of this report is unlimited.		
11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Office of Naval Research (Code 437) Information Systems Branch Department of the Navy	

13. ABSTRACT

This report describes research in information processing intended for application to man-machine communication processes. In order to allow a computer user to represent some problems more easily, we would like to let him name and define relations between information entities with which he deals, and then manipulate a large number of such relations stored by the computer. In order to make such a capability available, compact and easily-manipulated internal computer representations for large numbers of the various kinds of two-entity relations must be found. This report describes such a development for one basic logical form of relation, the transitive, anticommutative relation exemplified by "precedes", "includes", "is greater than", and similar phrases.

The report describes a method for storing directed graphs (of transitive anticommutative relations) in a list-structured computer memory. There are three important features of this representation method: A method of dividing a graph into a number of strata based on the lengths of paths in the graph, a recursive decomposition technique which produces successively less complex versions of the graph, and a recursive search technique which utilizes the stratification and decomposition to extract information from the graph. Some preliminary tests of the representation technique on graphs containing up to several hundred randomly chosen relations are described; the results of the test indicate that this representation may require less processing time and far less core storage than previously-used techniques when applied to large graphs.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer						
Programming						
Memory						
List Processing						
Symbol Manipulation						
Relational Data Base						
Man-Machine Communication						
Heuristic						
Problem Solving						
Concept						
Data Representation						
Directed Graphs						
Structure						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. **REPORT DATE:** Enter the date of the report as day, month, year, or month, year. If more than one date appears on the report, use date of publication.

7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.

8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. **AVAILABILITY LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through

_______________________________________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through

_______________________________________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through

_______________________________________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.

12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, or *(U)*.

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.