ON THE EFFICIENCY OF

CLIQUE DETECTION IN GRAPHS

by

ANTHONY HUNTER DIXON

B.Sc., University of British Columbia, 1968
M.Sc., University of British Columbia, 1970

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in the Department
of
COMPUTER SCIENCE

We accept this thesis as conforming to the
required standard

.....

.....

.....

THE UNIVERSITY OF BRITISH COLUMBIA

May, 1973

ABSTRACT

This thesis examines the devices employed by various algorithms to search for maximal complete subgraphs in graphs. Also known as cliques, in Chapter 1 these subgraphs are seen to play an important role in graph theory, information retrieval, Sociometry, logic design, and computational complexity.

The enumeration of cliques using the Harary-Ross, Bonner, Peay, and Bron-Kerbosch algorithms is discussed in Chapter 2. The Reduced Redundancy algorithm is introduced, and the performance of the five procedures is assessed using an alternative approach to empirical testing. Each of the algorithms is shown to generate a "derivation tree" for a given graph whose size can be used as a measure of its efficiency.

In Chapter 3, the possibility of exploiting vertex similarity is examined with a view to reducing the size of the derivation tree. As a consequence, algorithms are proposed for finding non-similar cliques. The concept of "complete subgraph equivalence" of vertices is introduced to develop a means for expressing the cliques of a graph as the Cartesian product of vertex subsets.

An algorithm for detecting the existence of a clique of order k in a certain class of k-partite graphs in polynomial time is proposed in Chapter 4. This class consists of all graphs reducible by the algorithm to k-partite graphs having at most two vertices per block of degree greater than 0. This

algorithm is shown to provide an efficient heuristic which can be used in a procedure for determining whether a well-formed formula is a tautology.

The thesis is concluded with an empirical analysis of the techniques employed by the enumeration algorithms of Chapter 2. In addition, the efficiency of the Clique Detection algorithm is compared with that of the Reduced Redundancy algorithm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I wish to thank my supervisor, Professor A. Mowshowitz, for his assistance both academically and financially in the development and presentation of this thesis, and also Professor E. Lawler for bringing to my attention the importance of the problem. Finally, I wish to thank my wife, Patty, for her assistance in the preparation of this manuscript.

CHAPTER 1: INTRODUCTION

1.1 DEFINITIONS AND NOTATION

The purpose of this thesis will be to examine procedures which search for maximal complete subgraphs of a graph. In particular the manner in which algorithms enumerate these subgraphs will be explored in order to discover why such algorithms have an exponential computation time. The problem of detecting the existence of a complete subgraph of a particular order will also be explored and an algorithm will be proposed which uses a different method from that of enumeration. This technique is instrumental in improving the efficiency of the procedure of clique detection, and for a particular class of graphs the algorithm can be shown to have a polynomial computation time.

First we shall provide a fairly complete list of definitions pertinent to the problem at hand. Unfortunately, there is no universally accepted terminology in graph theory and for this reason the author has chosen his definitions to be compatible where possible with the widely known text of Harary [43]. The definitions will also serve to introduce the notation to be used in the body of this text for the concepts defined.

DEFINITION 1.1: A graph G consists of a finite set V(G) of vertices together with a prescribed subset E(G) of unordered pairs of elements from V(G) called the edges of G. If (u,v) is

an edge of G then the vertices u and v are said to be adjacent.

DEFINITION 1.2: The degree , d(v) , of a vertex v in a graph G is the number of vertices in G adjacent to v.

DEFINITION 1.3: A labelling of a graph G with $|V(G)| = n$ vertices is an assignment of n distinguishing labels, one to each of the vertices in V(G).

DEFINITION 1.4: A k-partite graph $G(m_1, m_2, \ldots, m_k)$ is a graph whose vertices can be partitioned into k blocks $V_1, V_2, \ldots, V_k$ such that for any two vertices u and v in the same block (u,v) is not an edge of $G(m_1, m_2, \ldots, m_k)$. Given such a partition m_i denotes the number of vertices in block V_i.

DEFINITION 1.5: A complete k-partite graph $K(m_1, m_2, \ldots, m_k)$ is a k-partite graph such that for any two distinct blocks V_i, V_j and any vertex u in V_i, and any vertex v in V_j, (u,v) is an edge of $K(m_1, m_2, \ldots, m_k)$.

DEFINITION 1.6: The chromatic number, $\chi(G)$, of a graph G is the minimum number of blocks $V_1, V_2, \ldots, V_{\chi(G)}$ possible in any partition of V(G) such that for any two vertices u,v in the same block (u,v) is not an edge of G.

DEFINITION 1.7: The point independence number , $\beta_0(G)$, is the largest number of mutually non-adjacent vertices in a graph.

DEFINITION 1.8: The complement , $\bar{G}$, of a graph G is a graph such that $V(\bar{G}) = V(G)$ and (u,v) is an edge of $\bar{G}$ if and only if (u,v) is not an edge of G and $u \neq v$.

DEFINITION 1.9: Let the vertices of G be labelled 1 through n where $|V(G)|=n$. The adjacency matrix , A(G), of the graph G is a (0,1) matrix such that $a_{ij}=1$ if and only if (i,j) is an edge

3

of the labelled graph. The adjacency set ,A(v), of a vertex v is the set of vertices adjacent to v.

DEFINITION 1.10: A unicliqual vertex v of a graph G is one which belongs to exactly one clique of the graph (see definition 1.14).

DEFINITION 1.11: A subgraph H of a graph G is a graph with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ such that if (u,v) is an edge of H, then (u,v) is an edge of G.

DEFINITION 1.12 For any set of vertices $W \subseteq V(G)$, the induced subgraph G_W is the maximal subgraph of G with vertex set $V(G_W)=W$. That is, for any u,v in $V(G_W)$ (u,v) is an edge of G_W if and only if (u,v) is an edge of G.

DEFINITION 1.13: A complete subgraph of order k of a graph G is a subgraph defined on k vertices of V(G) for which any two vertices in the subgraph are adjacent. A triangle of G is a complete subgraph of order 3.

DEFINITION 1.14: A clique C of order k of a graph G is a complete subgraph of G for which there exists no vertex in V(G)-V(C) adjacent to all vertices in V(C). Cliques are therefore maximal complete subgraphs.

DEFINITION 1.15: An automorphism of a graph G is a permutation of the vertex set V(G) which preserves adjacencies.

DEFINITION 1.16: The collection of all automorphisms of a graph G forms a group called the automorphism group ,$\Gamma(G)$, of the graph G.

DEFINITION 1.17: Two vertices u,v of a graph G are similar if there exists an α in $\Gamma(G)$ such $\alpha u=v$.

DEFINITION 1.18: A graph G is point-symmetric if for any two

vertices u,v in V(G) there exists an α in $\Gamma(G)$ such that $\alpha u = v$.

DEFINITION 1.19: A graph G is line-symmetric if for any two edges $(u_1,v_1),(u_2,v_2)$ in E(G) there exists an α in $\Gamma(G)$ such that either $\alpha u_1 = u_2$ and $\alpha v_1 = v_2$ or $\alpha u_1 = v_2$ and $\alpha v_1 = u_2$.

DEFINITION 1.20: An algorithm which operates on a graph is considered efficient if the computation time and storage requirements can be expressed as functions of n bounded above by a polynomial in n, where n is the number of vertices in the graph.

DEFINITION 1.21: The Hadamard product , AXB, of two matrices A and B is the matrix C where $c_{ij} = a_{ij} \cdot b_{ij}$.

1.2 INTRODUCTORY REMARKS

The focal point of this thesis is the detection of cliques in graphs. The importance of this topic from both a graph theoretic and an applications standpoint will manifest itself in the historical survey of the next section.

There are several variants or special cases of this subject which can be considered. We will confine ourselves to a study of the following four problems:

1.) The enumeration of cliques of a graph;

2.) Detection of the largest clique of a graph;

3.) Determination of the non-similar cliques of a graph;

4.) Determination of cliques of a specific order.

The object in each case will be to study the special characteristics of the problem, and then to exploit these features to develop useful procedures for achieving the goal. In addition, an effort will be made to determine bounds on the efficiency of several such procedures and to seek insight into intrinsic properties of the methods used which might aid in estimating the best that one can expect from procedures designed to solve these problems.

1.3 HISTORICAL SURVEY

The study of methods for detecting maximal complete subgraphs originated principally with the search for an efficient and objective representation of the structure of social groups. Such groups can be modeled with a sociogram, a graph which characterizes the responses of individuals in a sociometric test which requires that each participant specify some subset of the group to which he wishes to belong (Moreno [62]). Pioneers in a mathematical treatment of this problem were Forsyth and Katz [33] who proposed representation of sociograms as matrices to which the elementary operations of row and column permutations could be applied to achieve some more desirable form in which the groupings of individuals could be more easily observed. Such a representation was subsequently refined by Luce and Perry [54], Festinger [31], and Luce [55]. A (0,1) matrix, effectively the adjacency matrix of the sociogram, was utilized, and the distance properties of a graph derived from the square, cube, and higher powers of the adjacency matrix were used as indicators in characterizing the structure of the group. Festinger observed that a unicliqual member i belonged to a clique of k persons if and only if the ith diagonal element in the cube of the adjacency matrix was equal to $(k-1)(k-2)$. It was therefore a simple matter to find the order of the clique to which an individual belonged, provided he was a member of only one clique. Weiss and Jacobsen [78] applied the techniques of Luce, Perry, and Festinger to analyze the relationships of individuals and their tasks in business organizations.

As a result of its sociological beginnings the word "clique" became synonymous with the graph theoretic notion of "maximal complete subgraph". The apparent usefulness of the graphical method of representation of sociometric data was suggested by the early results. However, investigators became aware of the general need for more powerful methods of manipulationg the sociogram and its associated matrix; in particular, of determining in an efficient but general manner the cliques suggested by the model. The importance of techniques for studying social groups thus motivated explcitation of a graph theoretic approach to the detection of cliques.

Because of the finiteness of the problem, there exists a naive, "brute force" method for finding the cliques of a graph with n vertices--merely examine each of the $\binom{n}{k}$ sets of k vertices for k=1,2,...,n. This clearly involves looking at $\sum_{k=1}^{n}\binom{n}{k}=2^n-1$ sets, an obviously intractable number for even moderately large graphs. Because sociograms could be very large, analysis by clique membership would be practicable only with the advent of much better detection procedures.

Early techniques depended primarily on a "bag of tricks" which incorporated knowledge of the structure of the graph induced by the particular application, and resorted to exhaustive search or estimation when simplification was no longer possible. In 1956 Harary and Ross [41] proposed a method whereby a graph was systemically reduced to components each of which contained at least one unicliqual vertex and for

which the observation of Festinger, previously mentioned, could be used to find it. The reduction procedure employed the following algorithm:

1.) Initially, let the graph G itself be the component under consideration.

2.) If v is a unicliqual vertex in a component under consideration, then the subgraph induced on v and those vertices adjacent to v is a clique. Define a new component for consideration by deleting from the current component v and all unicliqual vertices adjacent to v.

3.) If the component under consideration contains no unicliqual vertices, let v be a vertex belonging to a minimal number of triangles of that component. Define two new components for consideration:

a.) the subgraph induced on v together with those vertices adjacent to v, and

b.) the union of subgraphs induced on the set of vertices not adjacent to v together with their respective sets of adjacent vertices.

Since the union of these two components can be shown to be equal to the current component under consideration, delete it from the list of components to be considered.

4.) Choose a component from the list and go to 2 until the list is exhausted.

A procedure for improving the efficiency of the algorithm was also suggested by Harary and Ross by which a component

containing not more than three cliques could be completely processed at step 2. The Harary and Ross algorithm thus provided a more practical method than the naive algorithm for identifying cliques in sociometric data representable as a (0,1) matrix.

Subsequent researchers have pursued the development of techniques for the analysis of variations and generalizations of the model in an effort to include and obtain more information about the structure in a representation of a social group. By associating weights with each pair of vertices in a sociogram, the degree or strength of the relationship could be characterized (see for example Hubell [45]). By utilizing an adjustable threshold value, a hierarchy of graphs could be established, each graph being defined on the same set of vertices but consisting of only those edges whose weights exceeded the threshold value. Dorien [20], Johnson [47], and Boyle [7] have investigated the hierarchichal structure of groups by this method. In particular Peay [68] recently developed an algorithm for determining the hierarchichal clique structure of such a representation. A family of sets of vertices which we shall call "potential cliques" is generated from a graph G of order n and associated with a threshold value t.

1.) Initially V(G) is the only potential clique in the family. Denote by $w(v_i, v_j)$ the weight of edge (v_i, v_j) in E(G).

2.) If there exists an edge (v_i, v_j) in E(G) such that $w(v_i, v_j) < t$, and if there exists a potential clique C induced

on a subset of k vertices containing v_i and v_j then two new potential cliques C_1, C_2 are induced on $C-\{v_i\}$ and $C-\{v_j\}$.

3.) C is deleted from the family and C_1 and C_2 are added to it provided neither is contained in some current member of the family.

4.) When it is the case that $w(v_i,v_j) \geq t$ for any pair $(v_i\ v_j)$ in any potential clique, then the family constitutes the set of cliques associated with the threshold value t.

It is clear that the effect of the algorithm is to successively refine sets of vertices which contain non-adjacent pairs until such refinement is no longer possible; hence such sets of vertices can be considered to induce "potential cliques". To use the algorithm to find maximal complete subgraphs, a threshold value of 1 is assumed.

Yet another important sociological property of groups that stimulated the study of cliques in graphs was the tendency for individuals to "cluster" into groups in such a way that members of a cluster retained a high degree of similarity, while different clusters characterized dissimilar properties (Davis [15]). Cluster theory also had applications outside of Sociometry. Abraham [1] has used clustering techniques to solve the problem of minimizing the number of interconnections of electrical assemblies, a problem also explored by Lawler [51], while Bonner [6] applied such methods to medical taxonomy problems. Bonner's efforts resulted in the design of an apparently efficient algorithm, so considered

because his method eliminated the need for comparing newly generated vertex subsets with previously generated sets for containment, a necessary part of the procedures employed by Harary and Ross or Peay.

As a consequence, Bonner's algorithm enjoyed some popularity and was used in comparative studies with more recently proposed algorithms. However, Augustson and Minker [5] showed this efficiency was often illusory since a large number of extraneous vertex subsets could be generated in certain cases.

Besides the application of clique detection to the interpretation of sociometric data, the most recent widely explored application is to the problem of information retrieval. Early developments in the area of document retrieval were made by Meetham [57] while Abraham [2,3] applied such techniques to the problem of thesaurus construction. In addition, the previously cited work of Bonner was also an application of clique detection methods to information retrieval problems.

More recently, Gotlieb and Kumar [36] used clustering techniques to represent the degree of semantic association between index terms used in the classification of documents. The thesaurus problem was further explored by Augustson and Minker who employed a new algorithm developed by Bierstone which was shown by empirical methods to be better than that of Bonner in many cases. Bierstone's algorithm was compared empirically by Mulligan [64] with two recently developed

algorithms for clique enumeration, one by Bron and Kerbosch [8] and the second by Corneil (see Mulligan [64]). From this study Mulligan concluded the Bron-Kerbosch algorithm to be superior. The problems and techniques of comparing some of these algorithms will be discussed further in the next chapter.

Several important graph theoretical problems are related to the detection of maximal complete subgraphs. It is well known (see for example Nordhaus [67]) that the point independence number of a graph G is equal to the order of the maximal clique in G, its complement. In addition, the chromatic number of G is equal to the minimum number of independent cliques in G. At present there is no known efficient means of computing either of these graphical invariants. A procedure for determining either number would provide an important tool in pursuing the host of problems (see for example [14,19,26]) that exist in chromatic graph theory, and hence serves to emphasize the importance of studying complete subgraphs.

As a result the literature abounds with a variety of results relating to the existence of complete subgraphs in graphs. As an example, one may consider the celebrated problem of Ramsey [70] concerning the smallest number of vertices that a graph may have and contain either a complete subgraph of order m, or an independent set of k vertices. The determination of such a number, r(m,k), is an unsolved problem for general m and k, although the published results include

the calculation of specific values, existence theorems, and bounds [21,29,35,37,38,48]. It is evident that an efficient clique detection procedure would provide a useful tool by providing a faster way of examining graphs for their complete subgraphs.

Perhaps the results of extremal graph theory, pioneered by Turan[76,75], contributed most directly to the clique detection problem.

In 1965, Moon and Moser [60] verified by direct methods the maximum number of cliques possible in a graph, a result earlier established by Erdos [27] through an inductive argument:

The maximum number of cliques in a graph with n vertices is:

a.) $3^{n/3}$ if $n = 0 \bmod 3$

b.) $4 \cdot 3^{(n-4)/3}$ if $n = 1 \bmod 3$

c.) $2 \cdot 3^{(n-2)/3}$ if $n = 2 \bmod 3$.

It was also shown that the graphs which contain the maximum number of cliques were:

a.) the complete $\frac{n}{3}$-partite graph $K(3,3,\ldots,3)$ if $n = 0 \bmod 3$

b.) the complete $\frac{(n-1)}{3}$-partite graph $K(3,3,\ldots,4)$ if $n = 1 \bmod 3$

c.) the complete $\frac{(n+1)}{3}$-partite graph $K(3,3,\ldots,2)$ if $n = 2 \bmod 3$.

These graphs shall henceforth be referred to as Moon-Moser graphs.

It follows from this result that, as a function of the number of its vertices, a graph may contain an exponential number of cliques. Hence any algorithm which examines each clique at least once (ie. Sequential) may be expected to require an exponential amount of time to enumerate the cliques of a graph.

Although from a graph theoretic point of view this is a disheartening result, an examination of graphs which contain such numbers of cliques reveals that all cliques are of the same or nearly the same order. Further the cliques in such graphs appear to be homogeneously distributed over the vertices, each vertex belonging to the same or nearly the same (again exponential) number of cliques. From a practical point of view, the number of edges in the graphical models generated by the empirical data of the applications suggests that such conditions are unlikely to occur. The sparseness of the adjacency matrix of such graphs could therefore be used as a rough a priori test of the number of cliques a detection algorithm might be expected to find.

Cliques in graphs having a maximal number of cliques are all of the same size. Moon and Moser also showed that the number of different sizes of cliques in a graph with n vertices is bounded above by n-log(n). Erdos [30] improved their lower bound on this number by showing that it was bounded below by n-log(n)-H(n)-O(1), where H(n) denotes for some k the k-fold iterated logarithm, log log...log(n), and O(1) is an unspecified constant.

The results of Erdos, Moon and Moser suggest that an algorithm for the enumeration of cliques is an example of an exponential combinatorial process. There has recently been an effort in the theory of computation to establish a hierarchy of complexity classes of combinatorial algorithms, motivated by the lack of success in finding and proving efficient algorithms for a large number of important combinatorial processes. This work was pioneered by Cook [11] who showed that combinatorial problems can be expressed as language recognition problems. Using such a representation certain problems for which no efficient algorithms have yet been devised were shown to be equivalent in the sense that each was reducible to the problem of whether a well-formed formula was satisfiable. The class of problems so reducible has been expanded by Lawler [52,53] and Karp [49] and includes the clique detection problem. The principal result is that either there exists a polynomial bounded algorithm for each problem in the class, or for none of them. However, the nature of the problems strongly suggests that the latter case is in fact true.

The study of the detection of cliques may help to resolve this question since Mowshowitz [63] has shown that a well-formed formula with k clauses can be represented by a graph in such a way that the well-formed formula is a tautology if and only if there exists no complete subgraph of order k. This result is stated by Karp [49], and is implicit in Cook [11].

CHAPTER 2: ENUMERATION OF CLIQUES

2.1 INTRODUCTION

In order to gain insight into the complexity of the problem of clique enumeration, we shall examine the algorithms described in Chapter 1 in some detail. The technique evolved exploits the way in which the vertex subsets are determined during an iteration of each algorithm. A clique enumeration algorithm will then be proposed and shown to be more efficient than those previously examined.

The algorithms of Harary and Ross and Bonner were chosen because of their availability in the literature, their apparent differences of approach, and their frequency of citation as references in subsequent literature on the subject of cliques in graphs. In addition, the Harary and Ross algorithm is considered by this author to be the historical precedent for the development of clique enumeration algorithms. Peay's algorithm was chosen because it is comparatively recent, offers a conceptually simple approach to the problem and there appears to be no analysis of its efficiency. Although not yet readily available at the time of this documentation, the Bron-Kerbosch algorithm has been included because of its superiority over some of the previous methods as determined by Mulligan [64].

Although the algorithms cited employ apparently different techniques to achieve their goals, this difference is primarily one of detail, for an examination reveals the

following common features:

1.) Each algorithm refines or decomposes a previously determined vertex subset to obtain new vertex subsets each containing at least one clique of the original graph. A choice is made of a vertex from the initial vertex subset, and its adjacency properties are used to define the new subsets. The old vertex subset is subsequently removed from further consideration.

2.) Each algorithm has the property that every clique of the original graph is contained in exactly one of the possibly several vertex sets available for consideration at any stage of the algorithm.

3.) Each algorithm employs some device to avoid the pitfalls of multiply defining a clique or including as a clique some complete subgraph which is not maximal. Such a situation can occur whenever vertex subsets are generated which are properly contained in other vertex subsets, or which contain complete subgraphs maximal on that vertex subset but not on the original vertex set.

In the remainder of this chapter it will be seen that it is precisely these properties of the clique enumeration algorithms which profoundly affect the efficiency of such procedures.

2.2 MATHEMATICAL ANALYSIS OF ENUMERATION ALGORITHMS

To obtain some means of estimating analytically the computational time required by each algorithm, the computation may be divided into two parts. The first consists of determining the effort required for one iteration of the algorithm; that is, the time required to determine new vertex subsets from an old one. The second involves the determination of the number of iterations required to find all the cliques of the graph. We shall see that the number of iterations required is related to the number of vertex subsets generated during the execution of an enumeration algorithm A.

DEFINITION 2.1: A vertex subset W is directly derivable from a vertex set V using algorithm A if during some iteration in the execution of A, W is determined from V.

DEFINITION 2.2: A vertex subset W is derivable from a vertex subset V using algorithm A if there exist subsets $U_1, U_2, \ldots, U_j$ such that U_1 is directly derivable from V, U_{i+1} is directly from U_i for $i<j$, and W is directly derivable from U_j.

Since the set W is contained in the set V from which it was derived, using these definitions it is easy to see that derivability induces a partial ordering on the set of all vertex subsets generated by algorithm A during the course of enumerating the cliques, provided we add the stipulation that every subset is directly derivable from itself.

This partial ordering may be represented by a directed tree whose root represents the vertex set of the graph, and whose vertices represent the vertex subsets generated by

algorithm A. Vertex u, representing subset U, is connected by a directed edge to vertex w, representing subset W, if W is directly derivable from V. It is clear that this tree is dependent upon the enumeration algorithm used and the labelling of the graph, hence we shall always associate with any derivation tree a labelled graph G and an enumeration algorithm A.

DEFINITION 2.3: A vertex subset W will be said to be redundant if it is properly contained in some vertex subset V from which it was not derived.

It is interesting to observe that the behavior of clique enumeration algorithms to be described subsequently can be compared to a tree searching procedure, the tree in this case being the derivation tree of a graph G as determined by an algorithm A. The determination of methods for minimizing the development of redundant nodes in the determination of the derivation tree may then be likened to the problem of finding suitable tree pruning techniques. This similarity has also been noted by Mulligan [64], while Bron and Kerbosch have explcited it in their algorithm which used a "branch and bound" technique on the derivation tree.

To determine the computational effort required during a single iteration of each enumeration algorithm, a technique developed by Corneil [12, APPENDIX A] will be used. The tasks performed within each algorithm will be grouped into blocks. Each block will be defined from a set of basic operation types determined by implementation considerations. These operation

types and their associated time constants are given in TABLE 1 of APPENDIX A.

Most of the instruction types are self-explanatory. We shall elaborate briefly, however, on the "push" and "pop" operations. The need for such operations arises from the indeterminate amount of storage required for saving partially determined vertex subsets. It will be seen to be most useful to save such subsets, if they are needed for some subsequent processing, on a push-down store. Such a data structure is easily implemented as a linked list with additional storage added on as required. The essence of the "push" operation is to obtain storage for the current vertex subset to be saved together with a link address pointing to the next data item in the store. The top item is always directly accessible through a pointer to the top of the store . The "pop" operation deletes the top data item of the store and resets the pointer to the store top so that it points to the next data item which thus becomes the new top of the store.

In most cases it is difficult to obtain an exact expression for the computation time of the algorithm under consideration. This is due primarily to difficulties in determining the number of vertex subsets having a particular number of vertices. A second factor complicating the determination of such an expression is the difficulty of determining the extent of search to satisfy some condition (such as the "first edge" in Peay's algorithm) during a particular iteration. For these reasons our principal goal

will be to determine the order (as a function of the number of vertices in the graph) of an iteration, and the number of vertices in the derivation tree.

In determining estimates of the required computation time for the enumeration of cliques by various methods, the primary graph to be considered will be the complete k-partite graph with m vertices per block, denoted $K(m,m,\ldots,m)$ or $K(m^k)$. The reason for such a choice is that every k-partite graph is a subgraph of $K(m^k)$ and the derivation tree of any other k-partite graph is smaller than the derivation tree of $K(m^k)$. As a consequence of this choice of graph for consideration it is possible to determine the number of vertices in its derivation tree using each algorithm.

2.3 ANALYSIS OF THE HARARY-ROSS ALGORITHM

As mentioned in the introduction of this chapter, the Harary-Ross algorithm was selected for consideration in a comparative study of some clique detection algorithms because of its historical precedent. It is interesting to note that despite its frequent reference in subsequent papers on clique detection examined by this author, no mention had been made of an error in the algorithm until Harary himself referred to its existence in a recent paper on the application of graph theory to the Social Sciences [44]. Harary observed that although the method found all the cliques, it also included some "other" subgraphs in the set of maximal complete subgraphs as well. These "other" subgraphs are in fact complete subgraphs which are not maximal and hence each is contained in some clique. Although there appears to be no subsequent attempt made to correct the problem, possibly due to the existence of more efficient algorithms by the time of the discovery of the error, a modification to correct the defect is fairly simple. When a complete subgraph has been discovered, determine if there exists a vertex adjacent to all vertices in that subgraph. If not, a clique has been found. Otherwise it is obvious that such a subgraph is not a clique and is therefore deleted from further consideration. This modification is included in the subsequent analysis.

In the interests of improving the efficiency of their algorithm, Harary and Ross modified the general procedure cited in the historical survey by defining a procedure for

determining whether or not the subgraph induced on the vertex set under consideration had at most three cliques. If so, then such a subgraph could be completely processed by a direct method (rather than the recursive method of the general procedure) and the cliques of the subgraph determined. However in most cases only one additional iteration of the general procedure is required to determine all the cliques of a subgraph containing at most three cliques. In addition such a scheme requires additional computation, namely determining whether a subgraph has at most three cliques, during each iteration of the general procedure thus increasing the overall computation time. For these reasons, the contribution to overall efficiency is small and for simplicity has not been included in the analysis or implementation of the Harary-Ross algorithm.

2.3.1 NOTATION FOR THE ALGORITHM

G: The subgraph currently under consideration.

V(G) The vertex subset of the subgraph G.

A: The adjacency matrix of the subgraph G.

r: An array such that r(i) contains the sum of all elements in row i of $B = A^2 \times A$, the Hadamard product of the adjacency matrix and its square.

d: An array containing the degrees of the vertices in the subgraph G.

m: A pointer to the vertex i in G with minimum row sum r(i).

n: The number of vertices in the vertex subset V(G).

2.3.2 THE ALGORITHM

STEP0: Initially place V(G) on the stack.

STEP1: Choose a vertex set V(G) from the stack of vertex subsets to be considered. If stack empty then go to step13.

STEP2: Compute the matrix product A^2XA where "X" is the Hadamard product.

STEP3: Let $B= A^2XA$. Compute the row sums $r(1),r(2),\ldots,r(n)$ of B as well as the degrees $d(1),d(2),\ldots,d(n)$ of the vertices in the subgraph G.

STEP4: Set i to 1 and m to 1.

STEP5: If $r(i)=d(i)\cdot(d(i)-1)$ then go to STEP10.

STEP6: if $r(i) < r(m)$ then set m to i.

STEP7: set i to i+1. If $i \leq n$ then go to STEP5.

STEP8: No unicliqual vertices exist. Therefore define two new vertex subsets $V(G_1)$ and $V(G_2)$ as follows. $V(G_1)$ is the set of all vertices adjacent to m, the vertex of minimum row sum r(m). $V(G_2)$ is the set of all vertices not adjacent to m, together with all vertices adjacent to at least one of these vertices not adjacent to m.

STEP9: Store $V(G_1)$ and $V(G_2)$ on the stack of vertex sets to be considered and go to STEP1.

STEP10: A unicliqual vertex i has been found. Compute the intersection of all rows of the adjacency matrix of the original graph corresponding to vertices adjacent to i.

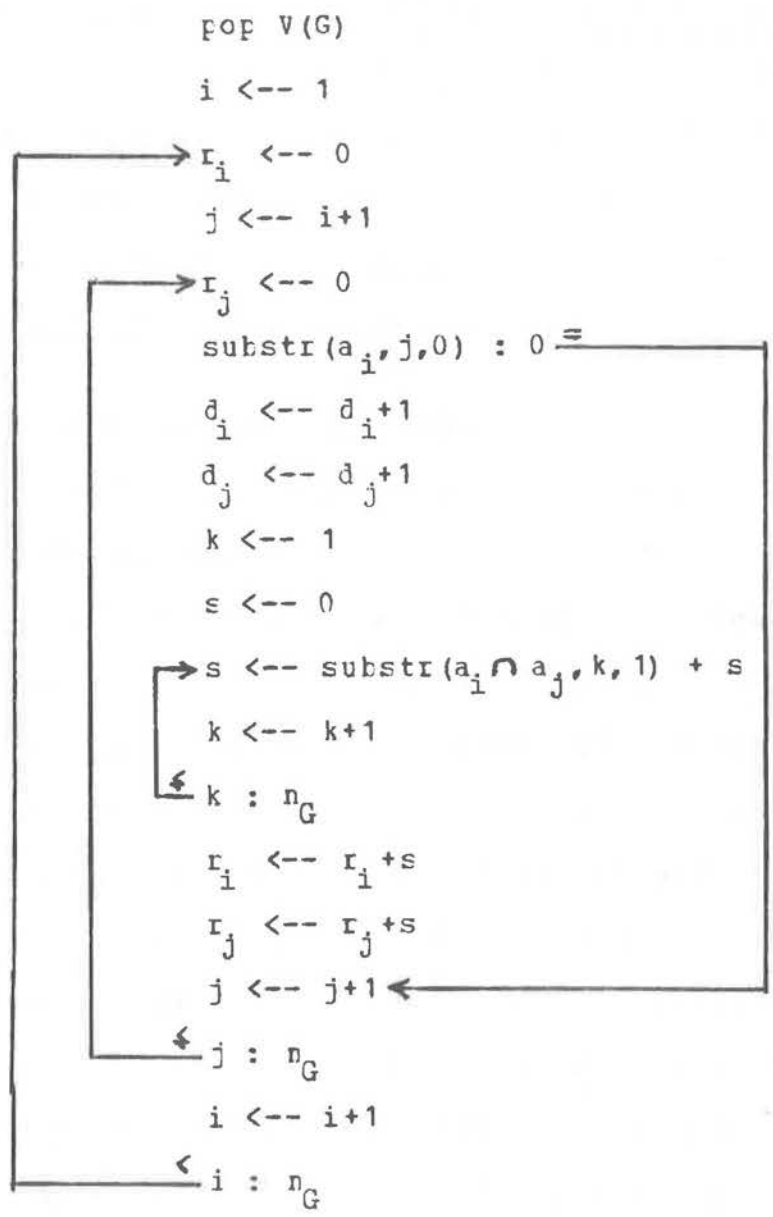
STEP11: If the result of STEP10 yields an empty set then the complete subgraph determined by i is maximal. Hence print i and the set of vertices adjacent to i.

STEP12: Delete from V(G) vertex i and all vertices adjacent to i which are also unicliqual. Place V(G) on the stack of vertex sets to be considered and go to STEP1.

STEP13: All vertex sets have been processed. Therefore stop.

The tasks to be performed by the Harary-Ross algorithm may be logically divided into four blocks. The function of block 1 is to compute the degrees of the vertices of the graph and to compute $[A(G)]^2 \times A(G)$ and determine the row sums of this matrix. Block 2 uses this information to search for a unicliqual vertex by finding a vertex which satisfies the relationship $r = d \cdot (d-1)$ where r is the corresponding row sum, and d the degree of the vertex. If such a vertex is not found, two new subgraphs are determined in block 3, one of which is returned to block 1, the other saved for further processing. Otherwise we proceed to block 4 to search for a complete subgraph of the graph G_1 induced on the unicliqual vertex v and those vertices to which it is adjacent. If the complete subgraph is a clique it is printed. All unicliqual vertices in the discovered complete subgraph are deleted from $V(G_1)$ and the subgraph G_1 is returned to block 1 for further processing.

BLOCK1



The parameter n_G is equal to the number of vertices of the vertex set V(G), and a(i) denotes the adjacency set of vertex i in G.

BLOCK1 dominates the computation in the Harary-Ross algorithm; the other blocks of the procedure will be seen to depend linearly on n_G. For this reason we defer their description until we have further examined BLOCK1.

In general it is difficult to obtain an explicit expression for the computational effort required by the Harary-Ross algorithm. This is due to difficulty in determining the number of times branch conditions are satisfied . It is also difficult, even for an arbitrary complete k-partite graph, to determine the number of vertex subsets that are generated with a particular distribution of vertices over the blocks. Instead, we shall consider the behavior of the Harary-Ross algorithm when finding the cliques of $K(3^k)$. A similar procedure could also be adopted for each of the other two types of Moon-Moser graphs. An example of the derivation tree for $K(3^k)$ is given in Fig. 2.2. Each node of the tree has been labelled according to the distribution of vertices among the blocks of the induced complete 3-partite subgraph which it represents.

Let G be an induced subgraph defined on a vertex subset obtained during the execution of the Harary-Ross algorithm, and suppose G has i_1 blocks with 1 vertex per block, i_2 blocks with 2 vertices per block, and i_3 blocks each containing 3 vertices. Necessarily it is the case that $i_1+i_2+i_3=k$ and $i_1+2i_2+3i_3=n$. The number of times that execution passes from line 6 to line 7 during the execution of block 1 is equal to the number of one's in the adjacency matrix of G above the diagonal. Since the original graph is complete k-partite every induced subgraph containing at least one vertex per block is also complete k- partite. The number of one's in G is therefore given by $f_G(i_1,i_2,i_3)$ where $f_G(i_1,i_2,i_3)=[(i_1+2i_2+3i_3)^2-(i_1+4i_2+9i_3)]$.

Hence steps 7 through 15 of block 1 will be performed $f_G(i_1,i_2,i_3)$ times and the computation time of block 1 during one iteration is given by $T_1(n_G)$ where

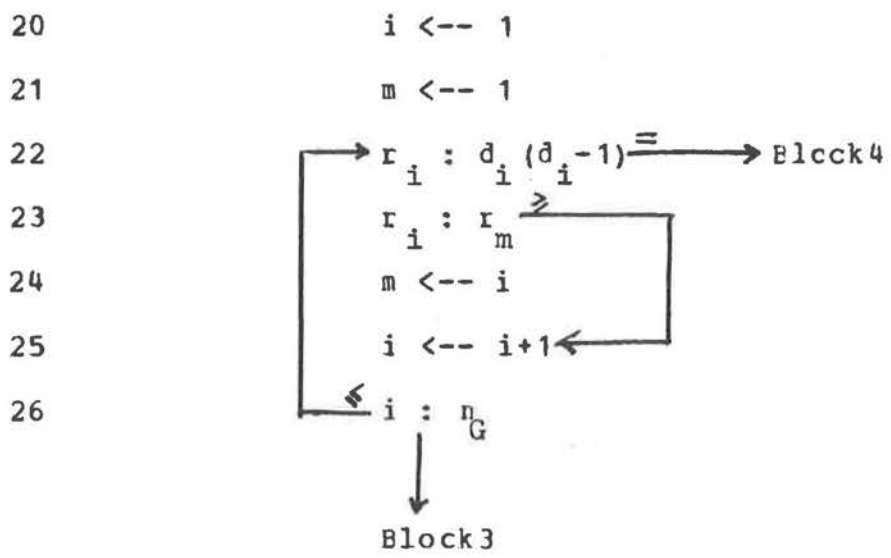
$$T_1(n_G) = C_1^1+n_G C_2^1+n_G(n_G-1)C_3^1+f_G(i_1,i_2,i_3)(C_4^1+n_G C_5^1)$$

with

$$C_1^1 = t_1+t_2$$
$$C_2^1 = 3t_1+2t_3$$
$$C_3^1 = 2t_1+t_3+2t_4+t_8$$
$$C_4^1 = 6t_1+4t_3$$
$$C_5^1 = 2t_1+2t_3+t_4+t_6+t_8$$

The constants t_i, $i=1,2,\ldots,10$, represent the cycle times for the various operations as specified in APPENDIX A.

BLOCK2



The computational time for BLOCK 2 is given by:

$$T_2(n) = C_1^2 + g_i C_2^2$$

where

$$C_1^2 = 2t_1$$

$$C_2^2 = \frac{3}{2}t_1 + 2t_3 + 3t_4 + t_5$$

It was assumed that line 24 which points to the minimal r(i) was executed one-half the time. The term g_i is equal to one less than the label of the first unicliqual vertex i encountered.

BLOCK3

```
27          V(G_1) <-- a_m ∩ V(G)
28          substr(V(G_1),m,1) <-- 1
29          i <-- 1
30          V(G_2) <--  -V(G_1)
31          i <-- i+1
31      --> substr(V(G_1),i,1) : 0 (≠)
32          V(G_2) <-- V(G_2) ⊔ a_i
33          i <-- i+1 <--
34      (≤) i : n_G
35          push V(G_1),V(G_2)
```

The computation for one iteration of block 3 is

$$T_3(n_G) = C_1^3 + (n_G - d_G(i))C_2^3 + n\ C_3^3, \text{ where}$$

$$C_1^3 = 4t_1 + 2t_2 + t_6 + t_7 + t_8$$

$$C_2^3 = t_1 + t_6$$

$$C_3^3 = t_1 + t_3 + 2t_4 + t_8$$

and $d_G(i)$ is the degree of vertex i in the subgraph induced on V(G).

BLOCK4

```
36                  C <-- a_i
37                  substr(C,i,1) <-- 1
38                  T <-- T∪-T
39                  k <-- 1
40              →   substr(C,k,1) : 0 ≠
41                  T <-- T∩a_k
42                  k <-- k+1 ←
43              ≤   k : n_G
44                  T : 0 ≠
45                  print C
46                  substr(V(G),i,1) <-- 0 ←
47                  j <-- j+1
48              →   substr(a_i,j,1) : 0 ≠
49                  r_i : r_j ≠ →
50                  substr(VG),j,1) <-- 0
51                  j <-- j+1 ←
52              ≤   j : n_G
53                  push V(G)
```

The computation time for BLOCK4 is

$$T_4(n_G) = C_1^{\bullet} + d_G(i)C_2^{\bullet} + n_G C_3^{\bullet} + hC_4^{\bullet} - iC_5^{\bullet},$$

the constants being given by

$$C_1^{\bullet} = 6t_1 + t_2 + t_3 + t_4 + 2t_8 + t_{10}$$

$$C_2^{\bullet} = 2t_1 + t_3 - t_4 + t_6$$

$$C_3^{\bullet} = 3t_4 + t_8$$

$$C_4^{\bullet} = t_1 + t_8$$

$$C_5^{\bullet} = t_1 + t_3 + 2t_4 + t_8$$

and h is the number of unicliqual vertices in the clique.

For $K(3^k)$ the next section will show that there are $\frac{1}{2}(3^k-3)$ components which do not have unicliqual vertices; hence the value of g_i in block 2 for each of these vertex sets is n_G, and therefore the maximum value for $T_2(n_G)$ is $C_1^2+C_2^2 n_G$.

For blocks 3 and 4 $d_G(i)$ and h are also bounded by n_G and hence $T_3(n_G)$ and $T_4(n_G)$ are also linear polynomials with respect to n_G. $T_1(n_G)$ is thus clearly the dominant term in the computation time for one iteration of the Harary-Ross algorithm and is a polynomial of order n^3.

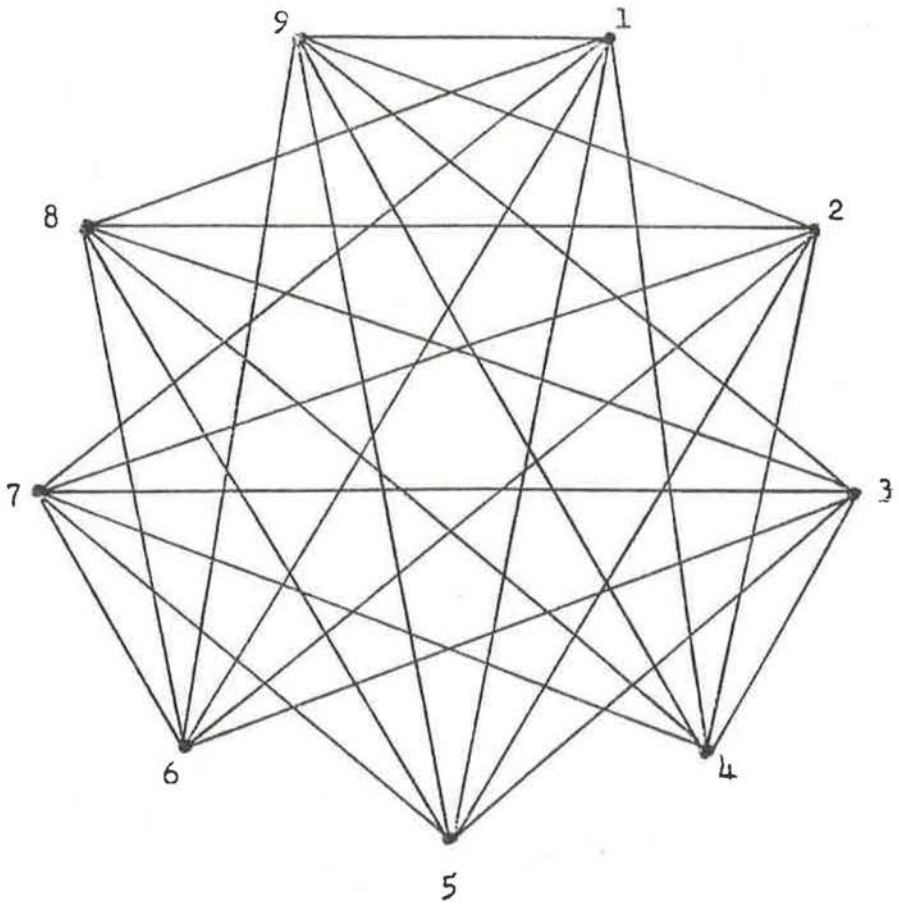
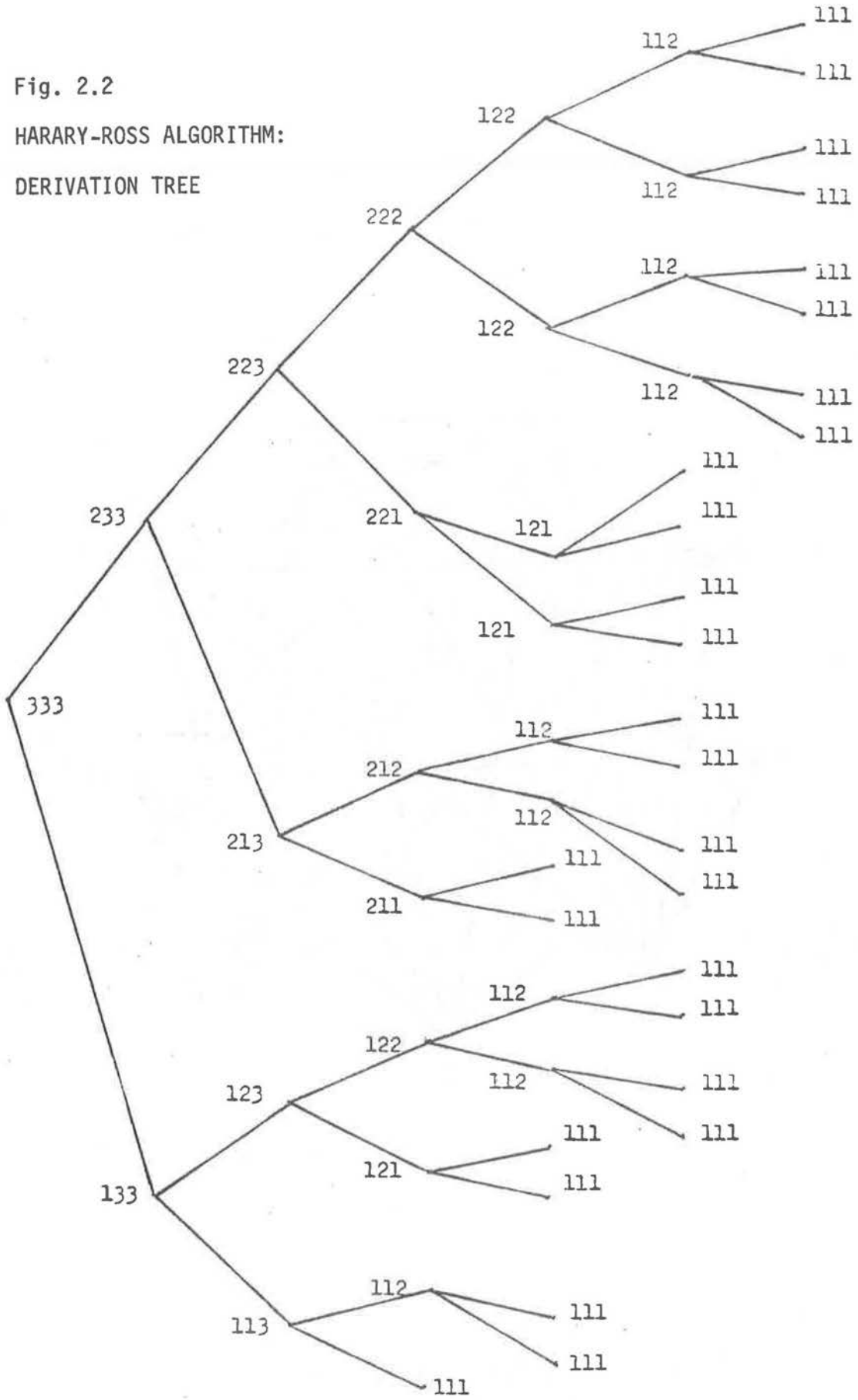


Fig. 2.1: K(3, 3, 3)

Fig. 2.2

HARARY-ROSS ALGORITHM:

DERIVATION TREE



2.3.3 NUMBER OF VERTEX SUBSETS EXAMINED BY THE ALGORITHM

From an examination of the derivation tree of an arbitrary graph using the Harary-Ross algorithm (see for example Fig. 2.2) it is easy to determine the number of components or vertex sets that will be generated. This is because, given that the number of cliques in the graph is N, since the derivation tree from this algorithm is binary, the number of nodes is 2N-1. The nodes of the derivation tree of $K(3^k)$ can be separated into two parts according to the type of processing carried out on the component represented by that node of the tree. In particular, whenever a unicliqual vertex is found via block 2, the clique to which it belongs is determined in block 4 and the component corresponding to this clique is output rather than returned to the processing stack, which consists of vertex subsets yet to be examined further for cliques.

For example, if we examine Fig. 2.2, and consider the subgraph induced on a vertex set having one vertex in every vertex block but one (eg. (1,1,3), (1,1,2), (1,2,1), (2,1,1)) we see that all vertices in the block containing more than one vertex are unicliqual. A vertex would be chosen from this vertex block, and through the computation in block 4 of the algorithm a clique would be determined and a vertex set returned for further processing. In the case of the example such a vertex set would have the form (1,1,2) or (1,1,1).

For the general complete k-partite graph $K(3^k)$, among all vertex sets generated having unicliqual vertices, there is

only one vertex set of the form (1,1,1,...,1,3). This is because such a set is derivable only from previous sets having either one or three vertices per vertex block. Such a vertex set yields three cliques according to the sequence of derivations given in Fig. 2.3.

Of these three cliques one is reprocessed in block 1 as a consequence of the previous discussion. Of the remaining 3^k-3 cliques, all are derived from vertex sets having two unicliqual vertices in some block, one vertex in each of the remaining blocks, and hence again according to the previous argument one-half of these will be reprocessed one further time.

The purpose of this discussion has been to ascertain how many vertex sets of the $2\cdot 3^k-1$ generated are subject to processing in block 1 where the major portion of computation occurs. This number is thus $3^k-1+1+\frac{1}{2}(3^k-3) = \frac{3}{2}(3^k-1)$. In addition, the number of vertex sets containing unicliqual vertices, and therefore examined in block 4, is $2+\frac{1}{2}(3^k-3)$. Finally the number of vertex sets not containing a unicliqual vertex and therefore processed in block 3 is given by

$$(2\cdot 3^k - 1) - 3^k - (2 + \tfrac{1}{2}(3^k - 3)) = \tfrac{1}{2}(3^k - 3)$$

If we denote by $\bar{T}_1, \bar{T}_2, \bar{T}_3, \bar{T}_4$, estimates of the computation time for blocks 1,2,3,and 4 respectively then an estimate of the computation time is given by:

$$T_{appx} = (\bar{T}_1+\bar{T}_2)(\tfrac{3}{2}(3^k-1)) + \bar{T}_3(\tfrac{3}{2}(3^{k-1}-1)) + \bar{T}_4(\tfrac{1}{2}(3^k+1))$$

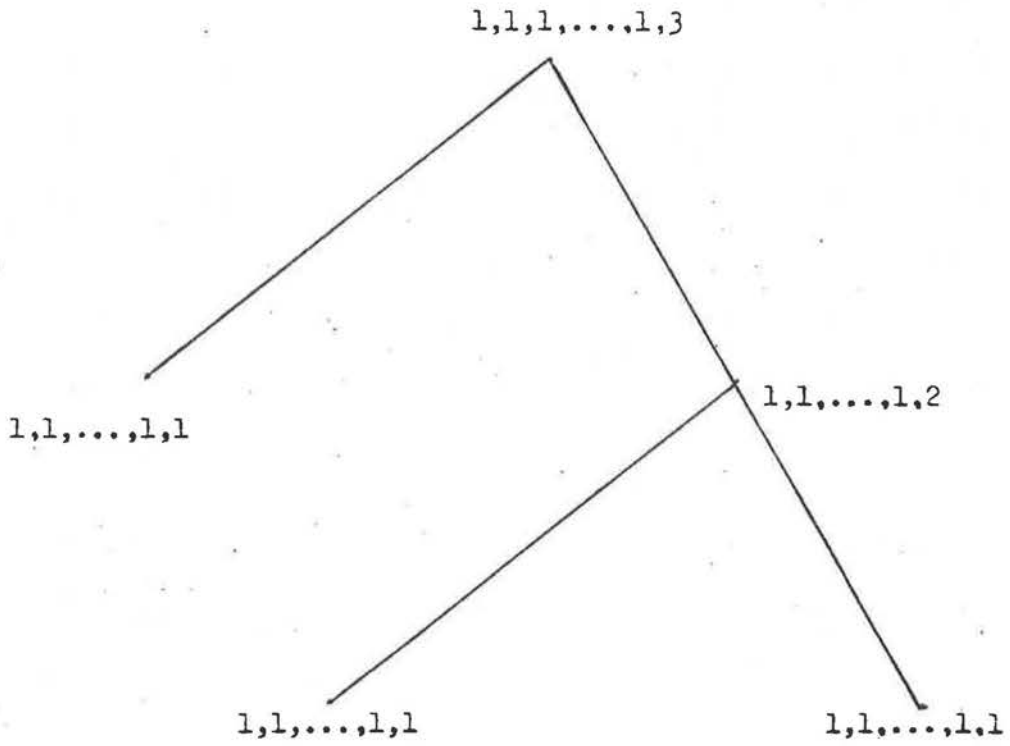


Fig. 2.3

SUBTREE OF DERIVATION TREE

2.3.4 STORAGE REQUIREMENTS

The Harary-Ross algorithm as defined in blocks 1 through 4 requires only one adjacency matrix be stored, that of the original graph. The appropriate sub-matrix is then determined during each iteration by keeping track of the appropriate vertices defining each induced subgraph. Also, since the r and d arrays are pertinent only to the induced subgraph currently under consideration, only one array of size n of each is required. As all other terms are counters or pointers the only other storage requirement is made by maintenance of a push-down store for keeping track of the vertex sets remaining to be processed.

At any iteration we usually define two vertex sets named $V(G_1)$ and $V(G_2)$ in the description of the algorithm. If we order their position on the push-down store so that $V(G_1)$ is always chosen first from the push-down store, then since no path in the derivation tree is of length greater than k, there cannot be more than k vertex sets waiting in the store. Each corresponds to the "other" vertex set paired with that vertex subset represented by a node lying on the path in the derivation tree. As an example consider the derivation of clique (258) in $K(3^k)$ as labelled in Fig. 2.1. The sequence of events is illustrated in Fig. 2.4.

For each pair of direct derivations, $V(G_1)$ corresponds to the "left" derivation, $V(G_2)$ to the "right" derivation. Before we can reach the vertex in the derivation tree labelled (23,456,789) we must have processed (1,456,789) since we have

arranged to do this first. Consequently all cliques containing vertex 1 have been determined and vertex set (1,456,789) or its derivatives no longer appear on the push-down store. A similar argument applies to (2,4,789) and (2,5,7). The only nodes remaining to be processed are (3,456,789), (2,6,789) and (2,5,9).

From a programming point of view it is most convenient as well as efficient to maintain lists of vertices as bit strings. Since both the vertex sets of the algorithm as well as the rows of the adjacency matrix are vertex lists it is clear that the storage requirements are given by 2n+C "integer units" of memory plus n(n+k) bits. An "integer unit" will depend on the storage conventions for integer representation on a particular system. For our purposes during implementation this is equal to a "half-word" or 16 bits.

The storage requirements for the Harary and Ross algorithm is thus n(n+k)+16(2n+C) bits where C is the number of pointers and counters required.

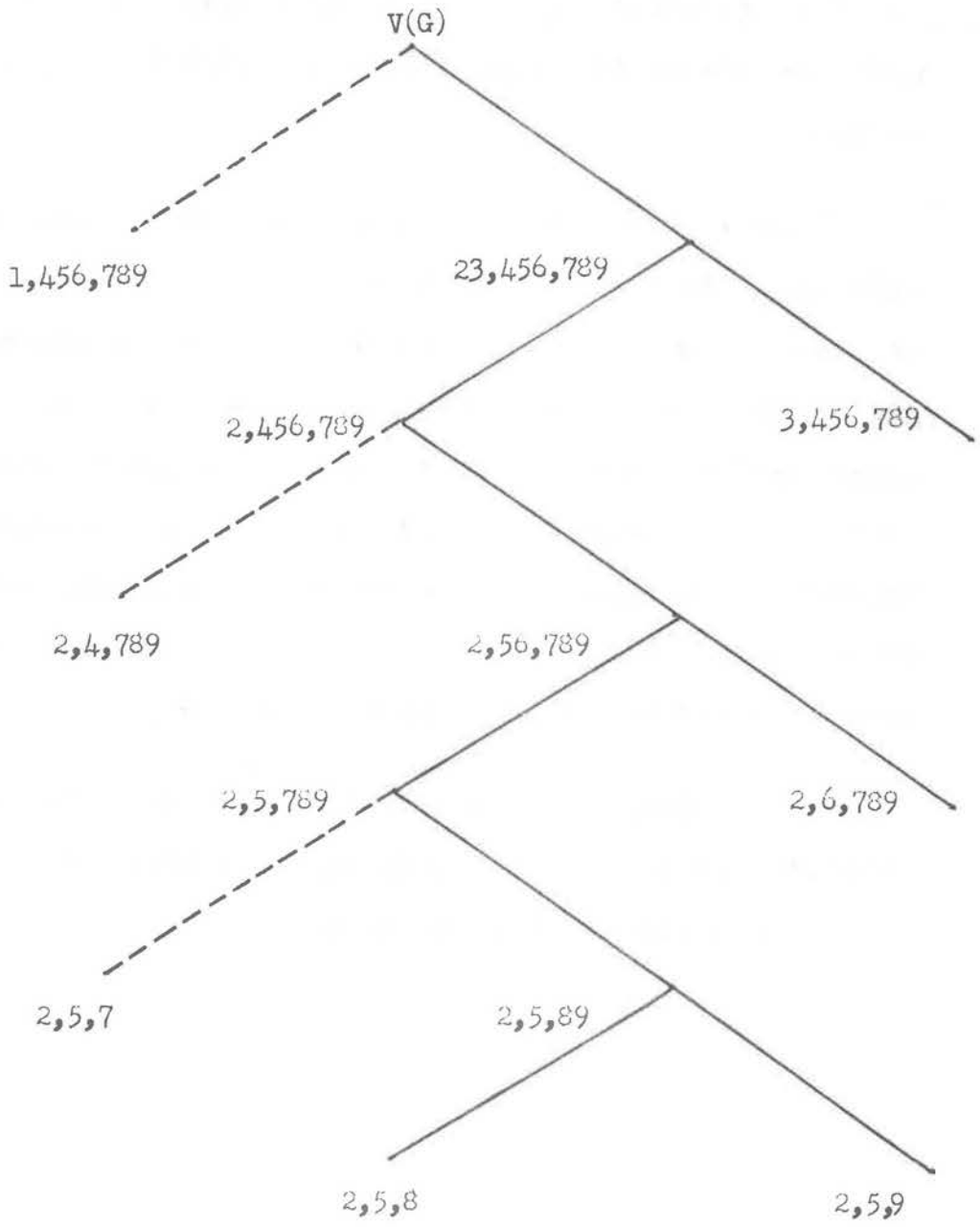


Fig. 2.4

A PATH OF THE DERIVATION TREE

2.4 ANALYSIS OF BONNER'S ALGORITHM

Bonner's algorithm has generated some interest among researchers wishing to employ the analysis of cliques in graphs to their particular application because of its apparent efficiency since no clique or vertex subset generated need be examined for containment in some previous component. This difficulty arose in the modified Harary-Ross algorithm previously described and is also inherent in Peay's algorithm, to be discussed next. In addition, Bonner's algorithm is interesting to examine in a comparative study of clique enumeration algorithms because it has been compared empirically with the efficiency of more recent algorithms.

The approach taken by Bonner is rather different from that of Harary and Ross or Peay in that it is a constructive procedure rather than one of reduction. The method employed is to build up the vertex sets of the cliques from a set of potential candidates, membership being determined by the adjacency properties associated with each vertex. We describe the steps of the algorithm as given by Bonner [6], including a minor correction noted by Augustson and Minker [5] in their discussion of the efficiency of the procedure.

The paper of Augustson and Minker showed that the efficiency of Bonner's algorithm may often be illusory because many complete subgraphs or components may be generated during the course of execution only to be discarded at some later stage. It was discovered that graphs containing several very large cliques and a few very small ones resulted in an

excessive amount of computation being performed on extraneous components which the algorithm would eventually delete. This generation of extra vertex subsets using Bonner's algorithm is also present in the enumeration of cliques of complete k-partite graphs, upon which we are focusing our discussion. We shall establish a generalization of observations made by Augustson and Minker which will then be used to determine the number of vertices in the derivation tree of $K(m^k)$ using Bonner's algorithm. We first, however, describe the procedure itself.

2.4.1 NOTATION

A : an array representing the set of objects in the complete subgraph to the present stage of calculation.

C : an array of potential candidates for increasing the size of the complete subgraph induced on vertices in A .

L_i : the last vertex of C to be considered for addition to the complete subgraph induced on A .

S : the adjacency matrix of the original graph.

2.4.2 BONNER'S ALGORITHM (Augustson, Minker[5])

STEP1: set i to 1, C_1 to V(G), A_1 to Φ , I_1 to 1.

STEP2: If L_i is not in set C_i then set I_i to I_i+1 and go to STEP5.

STEP3: Set C_{i-1} to $\{C_i \cap S(L_i)\}-\{L_i\}$ and A_{i-1} to $A_i \cup \{L_i\}$.

STEP4: set L_{i+1} to L_i+1, i tc i+1.

STEP5: If there is an element in C_i larger than L_i then go to STEP2.

STEP6: Set T to A_i. If $C_i=\Phi$ then A_i is a maximal complete subgraph. Else either A_i has been found before or it is nct maximal.

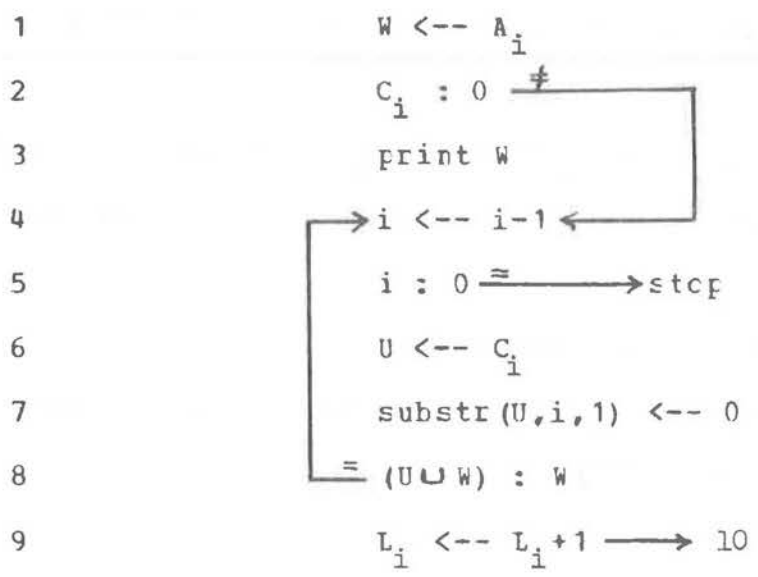
STEP7: Set i to i-1. If i=0 then stop.

STEP8: Set U to be the set cf all cbjects in C_i greater than L_i. If $U \subseteq T$ then go to STEP7.

STEP9: Set L_i to L_i+1 and go to STEP2.

The tasks performed by Bonner's algorithm have been divided into twc blocks. The function cf the steps performed in block 1 is to determine whether a discovered complete subgraph is maximal and to find the next component tc be processed. If one is fcund, ccntrcl is transferred to block 2 which constructs another complete subgraph returning the disccvered complete subgraph to block 1 for testing.

BLOCK 1



The computation time, T_1 for block 1 is given by

$$T_1 = C_1^1 + \delta t_{10} + hC_2^1$$

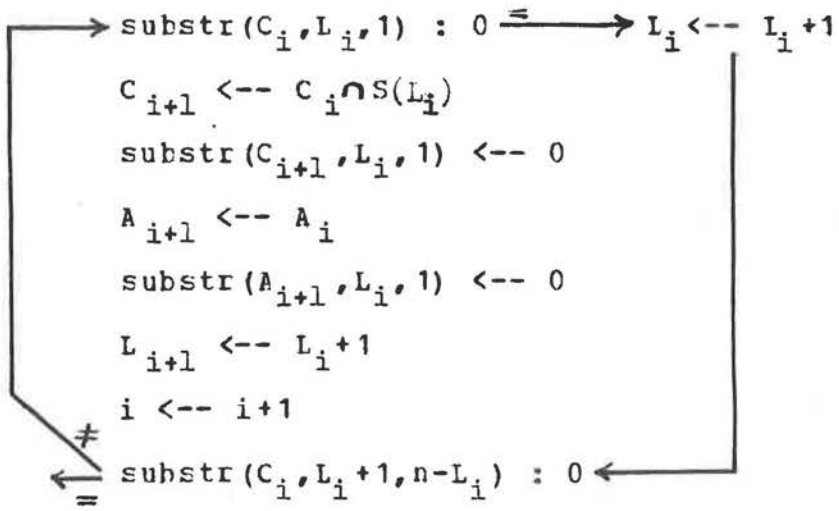
with constants being given by

$$C_1^1 = 2t_1 + t_3 + t_4$$

$$C_2^1 = 3t_1 + t_3 + 2t_4 + t_6 + t_8$$

and $\delta = 1$ if W is a clique, 0 otherwise, and $h \leq i$ is the first value of i for which $L_1, L_2, \ldots, L_i$ are the vertices of a complete subgraph contained in a clique not yet found.

BLOCK2



Every vertex subsequent to the original L_i is examined exactly once until there are no further vertices to be included. Therefore loop:17 to 10 is executed at most $n-L_i$ times for an arbitrary graph. For $K(m^k)$ there are at most $m(k-j)$ vertices in C_i, where j denotes the block to which vertex L_i belongs. For each value of i there are m-1 vertices not in C_i, namely the other vertices in the block. Since i is not incremented on such occasions loop:17 to 10 is executed m-1 times for each of the next k- j -1 blocks of vertices in C_i. If the graph is labelled such that vertices in block i have labels $(i-1)m+1, (i-1)m+2, \ldots, (i-1)m+m$, then $j = \left\lfloor \frac{L_i - 1}{k} \right\rfloor$. An upper bound on the time consumed in block 2 is given by T_2 defined as a function of L_i and n:

$$T_2(L_i, n) = \left(k - \left\lfloor \frac{L_i - 1}{k} \right\rfloor\right)\cdot\left(C_1^2 + (m-1)C_2^2 + mC_3^2\right)$$

where

$$C_1^2 = 6t_1 + 7t_3 + t_6 + 2t_8$$
$$C_2^2 = t_1 + t_3$$
$$C_3^2 = t_3 + 2t_4 + 2t_8$$

Since the value of h in block 1 is less than or equal to n while $k-\left\lfloor \frac{L_i-1}{k} \right\rfloor$ is maximized when L_i is in the first block, it is clear that the computation for one iteration of Bonner's algorithm is bounded by n = mk, the number of vertices in the graph $K(m^k)$.

2.4.3 NUMBER OF VERTEX SETS GENERATED

To determine the number of nodes in the derivation tree of Bonner's algorithm it is necessary first to establish the following result, a generalization of observations made by Augustson and Minker [5].

THEOREM 2.4: For $m \geq 2$ every complete subgraph of $K(m^k)$ is generated during the execution of Bonner's algorithm.

Proof: Using Bonner's notation the set A_i consists of a complete subgraph of order i defined on vertices labelled $L_1, L_2, \ldots, L_i$; the set C_i consists of all vertices adjacent to every vertex in A_i.

Suppose the vertices of block V_j in $V(K(m^k))$ are labelled $(j-1)m+1, (j-1)m+2, \ldots, (j-1)m+m$, for $m \geq 2$. If we perform the algorithm to obtain the "first" clique of $K(m^k)$ we obtain the following assignment to A_i and L_i for i=1,2,...,k:

$L_1 = 1$ $\quad A_1 = \{1\}$

$L_2 = m+1$ $\quad A_2 = \{1, m+1\}$

$L_3 = 2m+1$ $\quad A_3 = \{1, m+1, 2m+1\}$

....

$L_k = (k-1)m+1$ $\quad A_k = \{1, m+1, 2m+1, \ldots, (k-1)m+1\}$

Let U be a vertex subset of C_i consisting of all vertices with labels greater than L_i. Such a subset is not the vertex set of a complete subgraph unless there is at most one vertex in U because of the labelling of vertices in $K(m^k)$. Therefore, by execution of the algorithm, L_i is set to L_i+1 in step 8 and we return to step 2.

Since i is determined by the number of vertices in A_i when we entered step 6, and since every possible value of L_i from its initial one of $(i-1)m+1$ up to mk is adjacent to $L_1, L_2, \ldots, L_{i-1}$ and also contained in C_i, it is the case that a complete subgraph with vertex set given by A_i, $1 \leq i \leq k+1$, is generated where:

1.) $A_i = \{L_1, L_2, \ldots, L_i\}$,

2.) $(j-1)m+1 \leq L_i \leq mk$, $1 \leq j \leq i$

3.) $L_1 < L_2 < \ldots < L_i$. QED

We have thus established that for $m \geq 2$ every complete subgraph of $K(m^k)$ is generated during the execution of Bonner's algorithm. The special case $m=1$ corresponding to applying the procedure to a complete graph generates k complete subgraphs as described above in determining the first clique of the graph. Since every possible subset U of C_i is contained in A_k, clearly no return is ever made to step 2, so that the algorithm terminates after printing $A_k = \{1, 2, \ldots, k\}$.

The derivation tree for $K(3^3)$ is given Fig. 2.5 as an illustration of the vertex sets generated by Bonner's algorithm. From this one can clearly see the property of

Bonner's algorithm defined in theorem 2.4. As a result the number of components generated by Bonner's algorithm on $K(m^k)$ is given by the following:

THEOREM 2.5: The number of nodes in the derivation tree of $K(m^k)$ using Bonner's algorithm is $(1+m)^k$.

Proof: From Theorem 2.4 it is clear that every complete subgraph occurs as a node during some stage of execution. The number of complete subgraphs of order $i \leq k$ in $K(m^k)$ is equal to the number of ways of choosing i from k blocks $V_1, V_2, \ldots, V_k$, and then choosing 1 vertex from each of the i chosen blocks. This is $m^i \binom{k}{i}$. Hence the total number of complete subgraphs is $\sum_{i=1}^{k} m^i \binom{k}{i} = (1+m)^k - 1$. Since the root node of the derivation tree is not yet included this results in a total of $(1+m)^k$ nodes. QED.

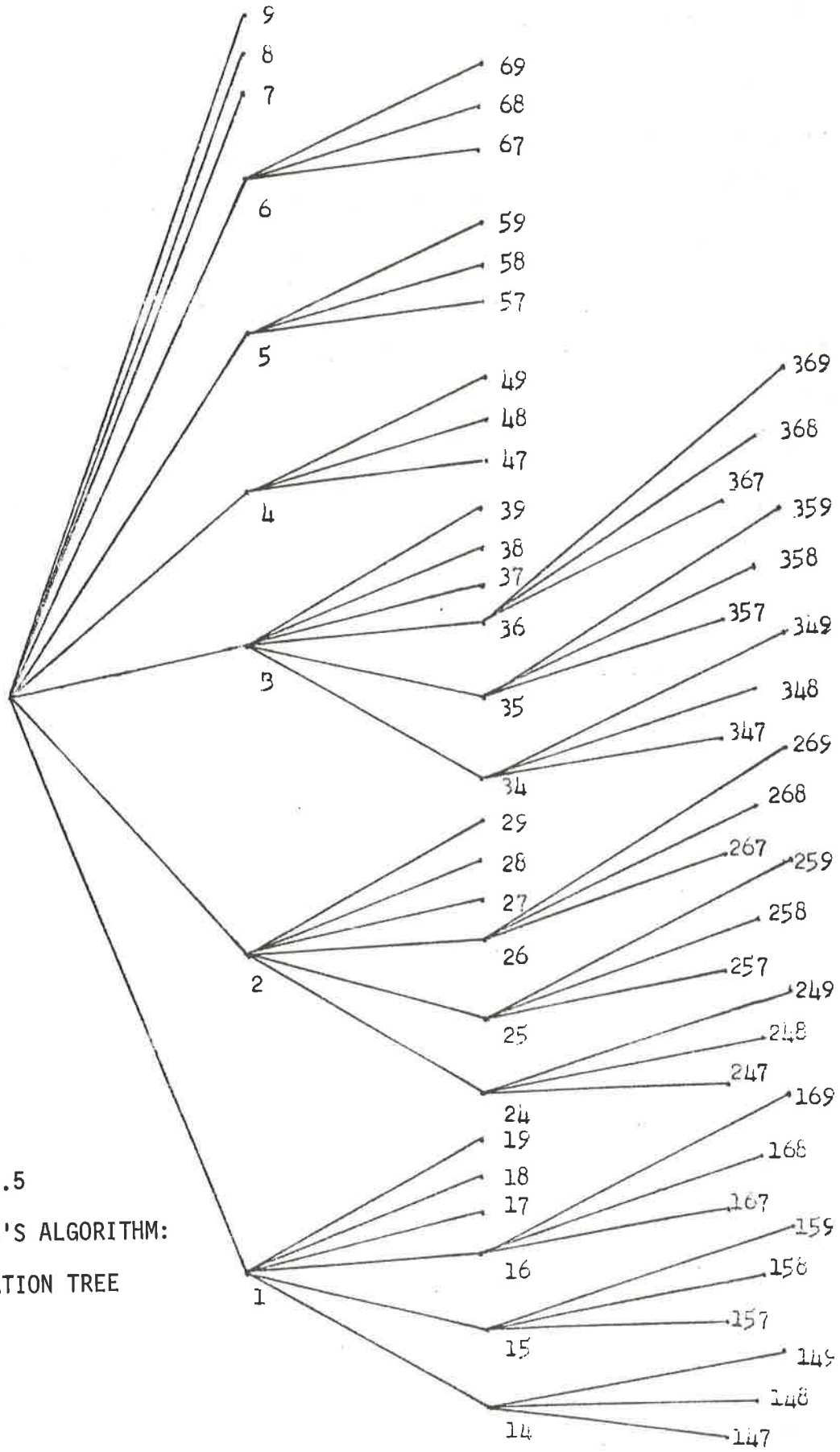


Fig. 2.5

BONNER'S ALGORITHM:

DERIVATION TREE

2.4.4 STORAGE REQUIREMENTS

The storage requirements for Bonner's algorithm are similar to those for the Harary and Ross algcrithm. Two arrays A and C of length n are required, each element corresponding to a vertex subset which can be represented as a bit string as can the rows of the adjacency matrix S. In addition an integer array of pointers L is required. Twc temporary bit strings T and U are also needed in addition to a counter i. Using the half-word of 16 bits as the integer unit, the storage requirements are: $3n^2 + 16(n+1) + 2n = 3n^2 + 18n + 16$ bits.

2.5 ANALYSIS OF PEAY'S ALGORITHM

The computation involved in the Harary-Ross algorithm was dominated by the computation of a matrix product and by the generation of a large number of components and their associated complete subgraphs, which were later deleted. Bonner's algorithm was also dominated by the generation of a number of superfluous components. Because Peay's algorithm generates only components which are essential to the final determination of all cliques, it is of interest as it may have a reasonably small derivation tree. However, the means by which Peay deletes non-essential components results in a large number of additional operations. Specifically, Peay compares each of two newly generated components to an ever growing list of vertex sets of cliques and subgraphs which are potential cliques. Thus, for a graph with an exponential number of cliques, as a function of the number of vertices, an exponential number of comparisons is required in addition to the time for generation. As will be seen, this off-sets to a considerable extent the time saved by avoiding the analysis of redundant vertex sets. For this reason we discuss here a modification to the algorithm which reduces the amount of storage required. The extent of this reduction is determined in our storage analysis. The procedure to be implemented for obtaining this improvement depends on ordering the selection of vertex subsets so as to obtain a development of "depth before breadth" of the derivation tree. The stack containing these vertex subsets is then altered to contain only those vertex sets which do not induce complete subgraphs. This

drastically reduces the size of the push-down store by eliminating the growing list of cliques previously being kept there. Instead, a test for clique membership is made, like that employed in our modification of the Harary-Ross algorithm. These modifications are included in our subsequent analysis.

Before proceeding, we should note however that the inefficiencies inherent in the algorithm as cited by Peay are a consequence of the application to which such a procedure was being put, namely the determination of a hierarchy of cliques in sociograms. As a rule the goal of a graphical treatment of such data is to assign the "individuals" to one or more of a few sets which it is hoped characterize the structure of the group. Hence the number of cliques in a social group as determined by such an analysis is small and therefore the difficulties of a possibly exponential number of cliques is not pertinent. As our treatment of clique detection algorithms is graph theoretic, we have not assumed any a priori information about the structure of the graph induced by its physical interpretation and must therefore be concerned with such problems.

2.5.1 NOTATION

V(G): the vertex set currently under consideration.

G: the subgraph induced on V(G)

n: the number of vertices in G.

M: the number of vertex sets in the stack.

A(i): adjacency set of vertex i.

$V(G_1), V(G_2)$: newly generated vertex sets.

G_1, G_2: the subgraphs induced on the new vertex sets.

2.5.2 THE ALGORITHM

STEP0: Initially place V(G) on the stack.

STEP1: Choose a vertex subset V(G) from the stack of vertex sets to be considered. If the stack is empty then stop.

STEP2: Find a pair of vertices v_i, v_j both in V(G) such that (v_i, v_j) is not an edge of the original graph. If no such pair exists then go to STEP5.

STEP3: define new vertex sets $V(G_1)=V(G)-\{v_i\}$, $V(G_2)=V(G)-\{v_j\}$.

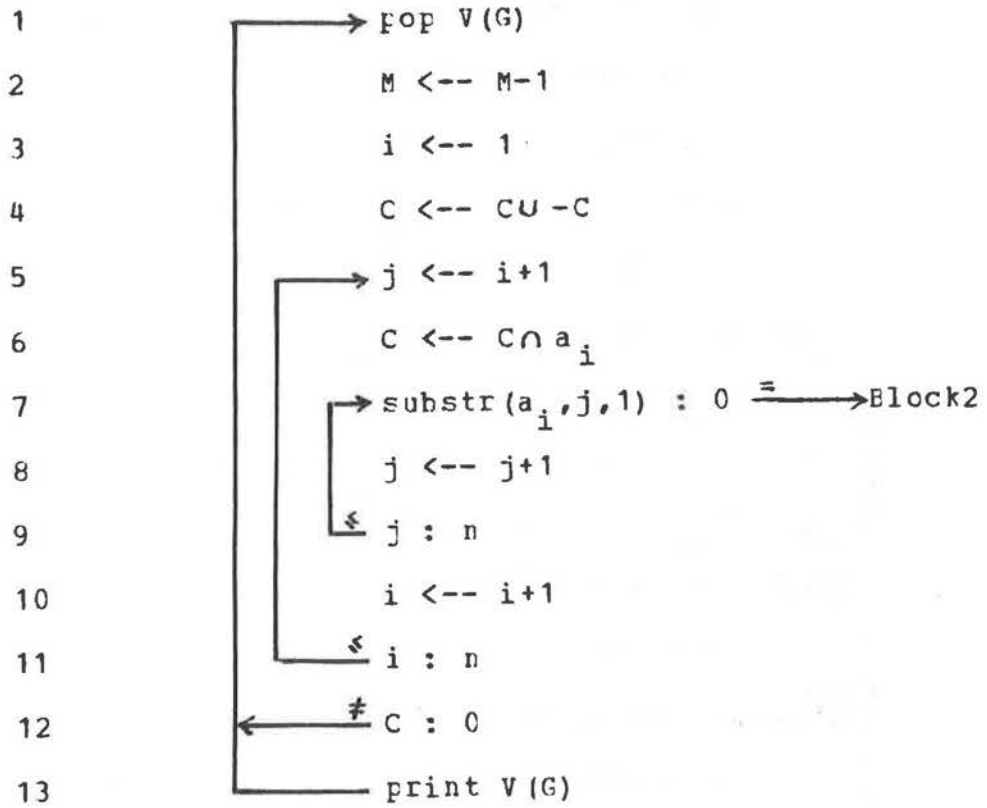
STEP4: For k=1,2, if $V(G_k)$ is not contained in vertex set currently on the stack then put $V(G_k)$ on the stack. Go to STEP1.

STEP5: V(G) induces a complete subgraph. If there exists no vertex in the original graph adjacent to all vertices in V(G) then print V(G) as a clique. In either case go to STEP1.

The tasks of Peay's algorithm can be logically grouped into two blocks. The function of block 1 is to examine the subgraph induced on a subset of the vertices of a graph G, in order to find a pair of non-adjacent vertices. If a pair is not found then a clique has been discovered and it is printed. If two vertices, say v and w, are not adjacent then control is passed to block 2 which defines two new vertex sets. Each is

saved for further processing provided it is not contained in some previously generated vertex set. Control returns to block 1 which chooses another vertex set for examination.

BLOCK1



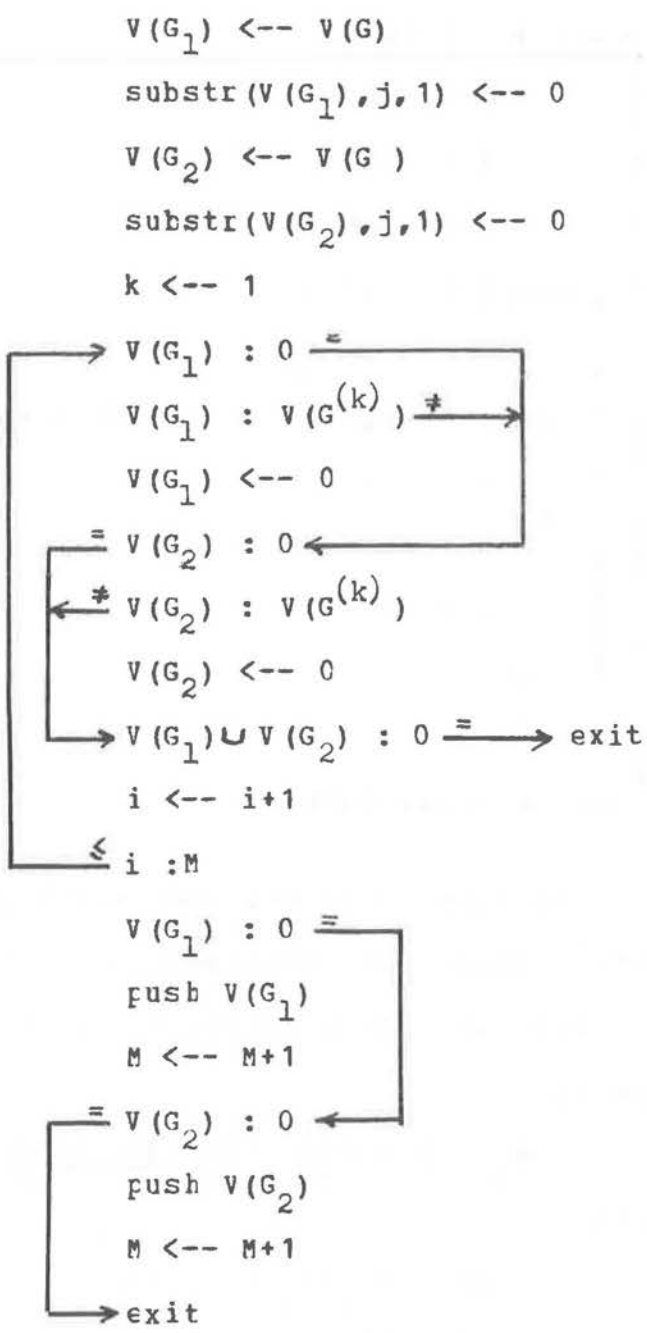
If G is in fact a clique, then all $\frac{n\cdot(n-1)}{2}$ ones in its adjacency matrix above the diagonal will be examined. The computation time of block 1 for one iteration is therefore bounded above by

$$T_1(n) = C_1^1 + n\,C_2^1 + \frac{n\cdot(n-1)}{2}C_3^1$$

with constants

$$C_1^1 = 3t_1 + t_2 + t_3 + t_4 + t_{10}$$
$$C_2^1 = 3t_1 + 2t_3 + t_4 + t_6$$
$$C_3^1 = t_1 + t_3 + 2t_4 + t_8$$

BLOCK2



The computation time is maximized when neither new vertex set is contained in some previous vertex set. When this occurs the time for BLOCK2 is $T_2 = C_1^2 + MC_2^2$ where

$$C_1^2 = 7t_1 + 2t_2 + 2t_3 + 2t_4 + 2t_8$$

$$C_2^2 = t_1 + t_3 + 6t_4 + t_6$$

and M is defined in section 2.5.1.

2.5.3 NUMBER OF VERTEX SETS GENERATED

The complete derivation tree for K(3,3,3) using Peay's algorithm is quite extensive. As will be seen this is due to the nature of the development of new vertex sets for consideration. In order to obtain an expression for the number of nodes in the derivation tree it will be convenient to consider the development which occurs during the processing of one block of vertices. That is, since Peay's algorithm determines two new components whenever a vertex pair is discovered which is not an edge of the graph, we shall consider all such pairs defined upon a single vertex block of G. Fig. 2.6 gives such a development for block $V_1 = \{1,2,3\}$ of K(3,3,3) as labelled in Fig. 2.1.

Examination of this sub-tree of the derivation tree reveals that three components are eventually generated with the property that each has exactly one vertex in block 1 and 3 vertices in each of the remaining two blocks. Since, for each of these sets, the one remaining vertex is adjacent to all other vertices in the vertex subset of that component it is evident that no further computation will involve that vertex. Thus the induced subgraph of each vertex subset is equivalent to K(3,3) with vertices labelled 4,5,6,7,8,9. The number of nodes generated is given by the sum of those determined during the generation of three components, K(1,3,3), from one component, K(3,3,3), and the number of nodes generated in the reduction of each K(1,3,3) (which is equivalent to the reduction of K(3,3)). Therefore if for $K(m^k)$ we can determine

the number of nodes created in generating m components $K(1, m^{k-1})$ we can obtain a recurrence relation for the number of nodes in the derivation tree of $K(m^k)$ using Peay's algorithm.

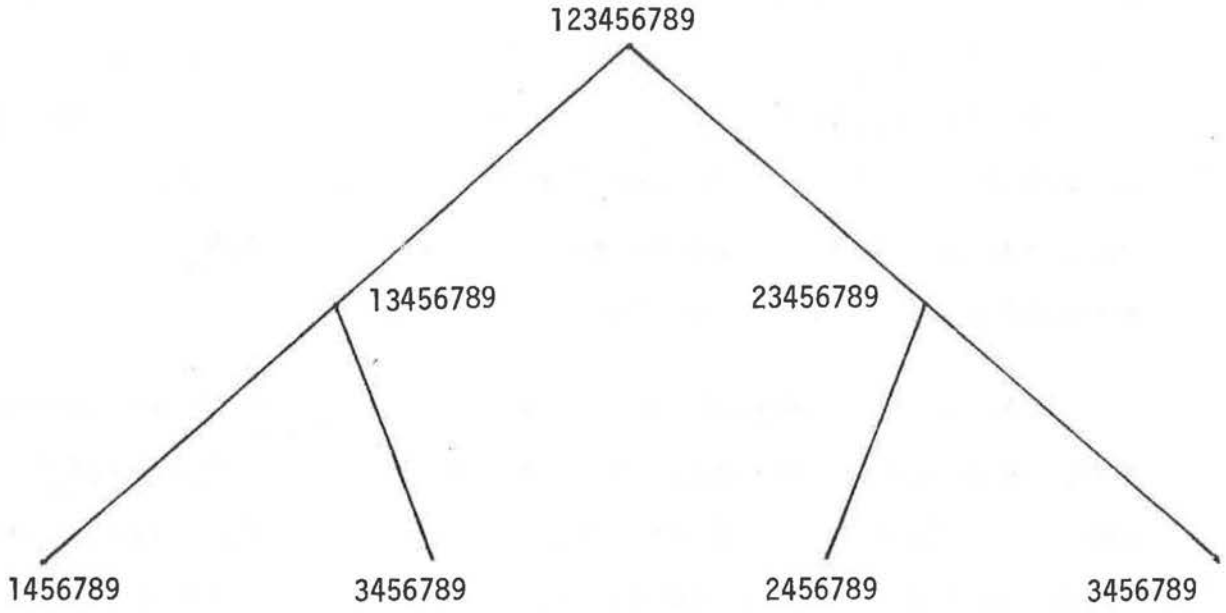


Fig. 2.6

SUBTREE OF THE DERIVATION TREE

THEOREM 2.6: The number of vertex sets generated by Peay's algorithm to find the cliques of $K(3^k)$ is $3(3^k)-2$.

Proof: Let $V_1, V_2, \ldots, V_k$ be the blocks of $K(m^k)$ and label the vertices in block V_j: $(j-1)m+1, (j-1)m+2, \ldots, (j-1)m+m$. Consider the sequence of derivations defined by processing the vertex pairs (non-edges) $(1,2), (1,3), \ldots, (1,m)$. The vertex sets derived starting from $(\{1,2,3,\ldots,m\}, V_2, \ldots, V_k)$ are respectively $(\{1,3,4,\ldots,m\}, V_2, \ldots, V_k), (\{1,4,5,\ldots,m\}, V_2, \ldots, V_k), \ldots, (\{1,m\}, V_2, \ldots, V_k)$, and finally $(\{1\}, V_2, \ldots, V_k)$. If $(\{1,i,i+1,\ldots,m\}, V_2, \ldots, V_k)$ is a typical vertex set from this sequence of derivations, two new sets, $(\{1,i+1,\ldots,m\}, V_2, \ldots, V_k)$ and $(\{i,i+1,\ldots,m\}, V_2, \ldots, V_k)$, are derived by separating vertices 1 and i.

The latter vertex set is deleted since it is contained in the previously determined set $(\{2,3,\ldots,m\}, V_2, \ldots, V_k)$. We therefore have a total of $2(m-1)$ vertex sets determined during this sequence of derivations, half of which are deleted, the remaining ones being those given above. This process is illustrated in Fig. 2.7.

The number of vertex sets considered in reducing one vertex block of $K(m^k)$ is given by $1+\sum_{i=1}^{m-1} 2i = 1+m(m-1)$.

Let a_k^m be the number of vertices in the derivation tree of $K(m^k)$ using Peay's algorithm. Since the reduction of one block of $V(G)$ yields m vertex sets whose processing is equivalent to that for $K(m^{k-1})$, the number of vertices is given by the recurrence relation $a_k^m = 1+m(m-2)+ma_{k-1}^m$ whose solution is: $a_k^m = C_1^m m^k + 1 + m(m-2)$. The complete derivation tree

for K(3,3) is given in Fig. 2.8 from which we obtain a_k^m with m=3. Solving for C_1^3 we get $C_1^3 = 3$ and $a_k^3 = 3(3^k)-2$. QED.

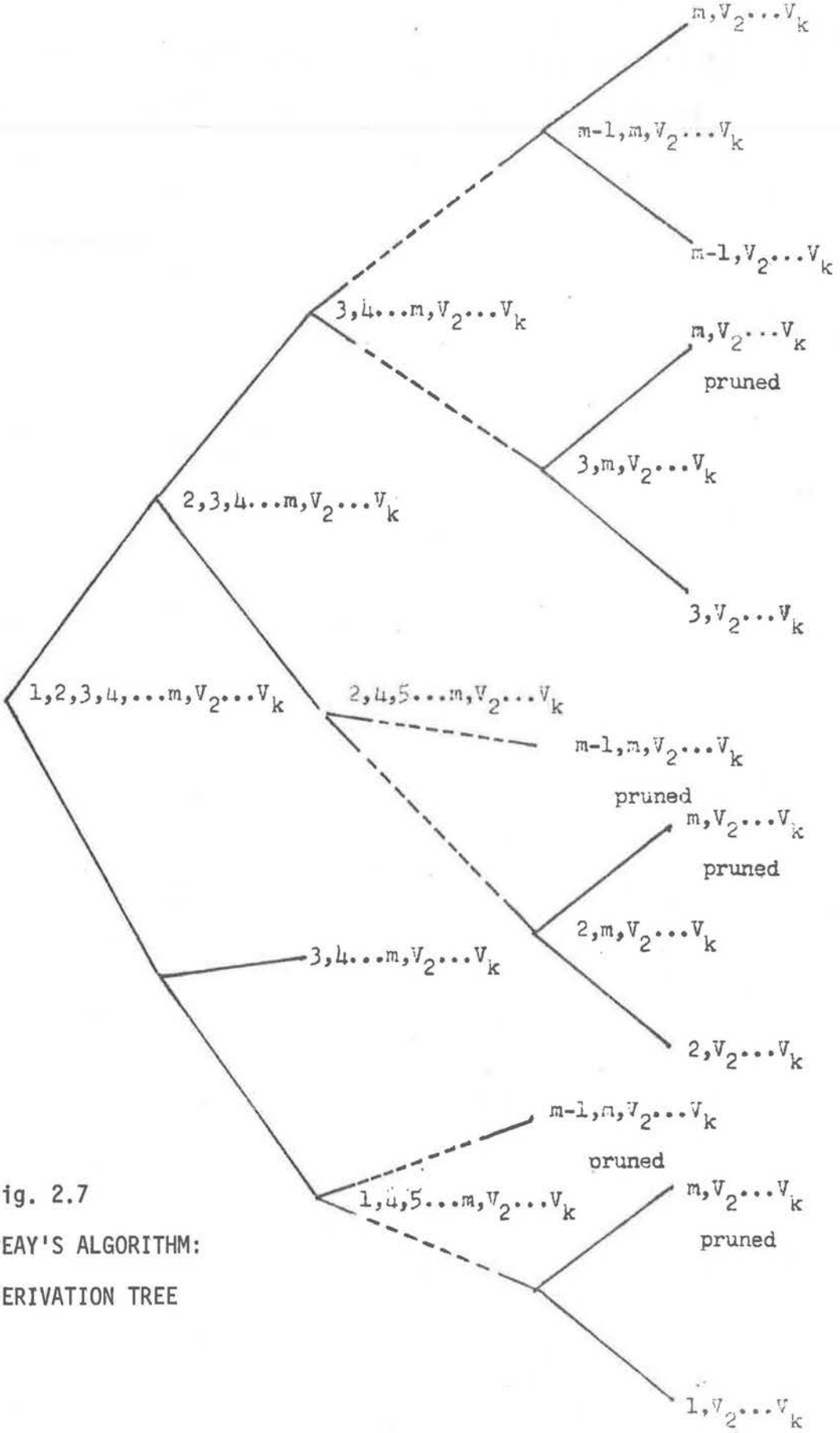


Fig. 2.7

PEAY'S ALGORITHM:

DERIVATION TREE

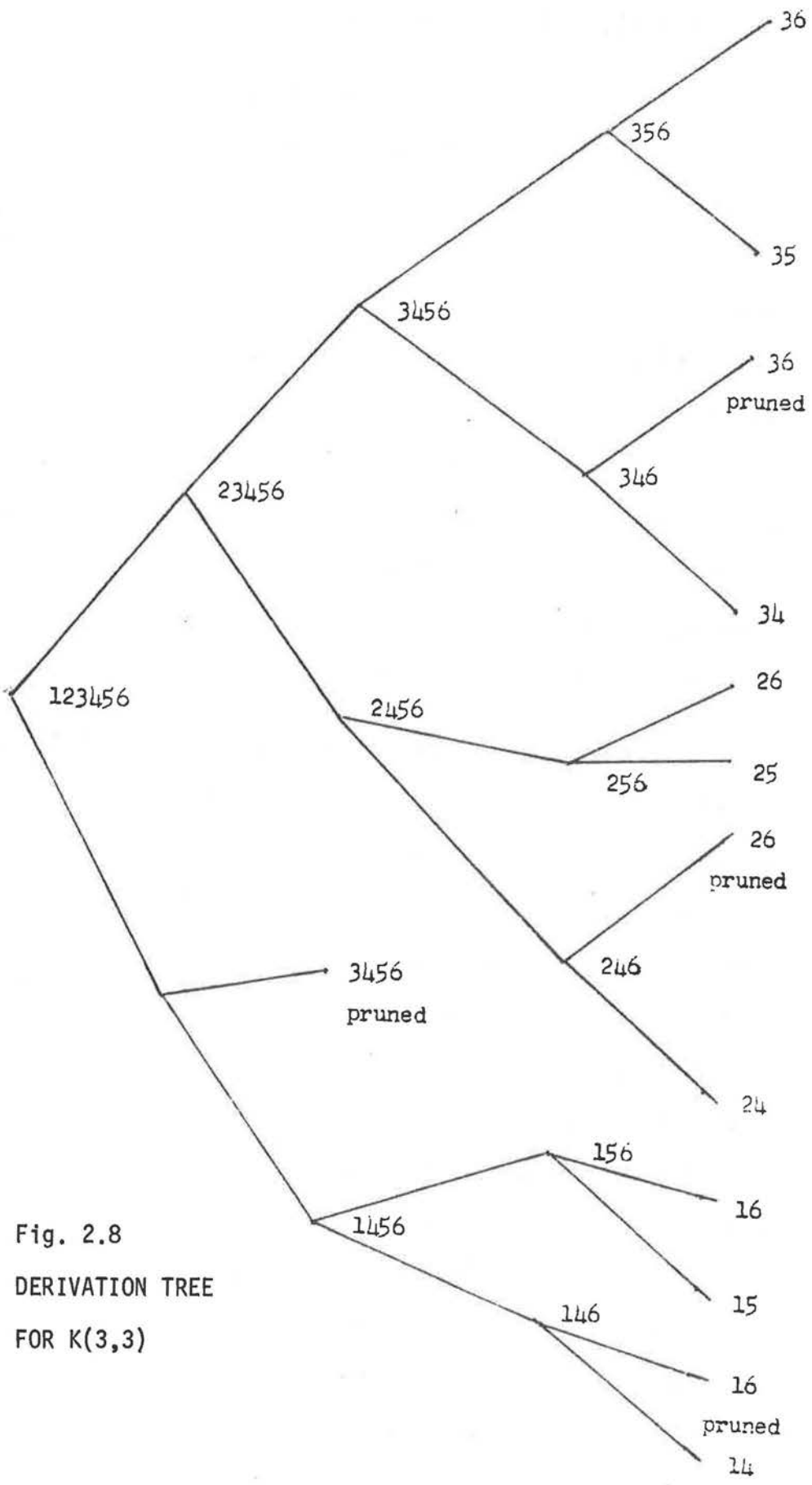


Fig. 2.8

DERIVATION TREE

FOR K(3,3)

2.5.4 STORAGE REQUIREMENTS

As has been observed, Peay's algorithm as originally defined required storage space for all new vertex sets generated during its execution. For a graph with an exponential number of cliques such a demand is not practicable. In our discussion we have described and analyzed a modification to the procedure which eliminates the need for maintaining in the stack the cliques as they are discovered. By developing each path in the derivation tree as far as possible, the number of nodes placed in the stack is never greater than the length of the path generated, each entry corresponding to the "other" vertex subset of the pair of vertex subsets developed at that stage.

Let V(G) be a vertex subset such that V(G) induces a complete k-partite graph having 1 vertex in i blocks and m vertices in (k-i) blocks. If we carry out a sequence of derivations by fixing one vertex in block i+1, say v and sequentially derive new vertex subsets from the set of non-edges $(v,w_1),(v,w_2),\ldots,(v,w_{k-1})$ then according to the argument presented in a discussion of the derivation tree for $K(m^k)$ developed by Peay's algorithm, two vertex sets will be added to the stack after such a sequence of derivations. The first consists of all vertices of the original vertex subset other than v. The length of the path in the derivation tree corresponding to this sequence is m-1, the number of vertices not adjacent ot v. Since there are k blocks in $K(m^k)$, the maximum length of any path is (m-1)k. However by choosing non-

edges as described, after the (m-1)i th node only one of the i+1 vertex subsets have been saved for further processing, hence the number of vertex sets on the stack is at most k+1.

The only other storage required is that for the adjacency matrix and a number of integer counters. Therefore, since each entry in the stack can be represented by a bit-string of length n, the total storage requirements are $n(k+1) + n^2 + 16C$ bits.

2.6 A NEW ENUMERATION ALGORITHM

The analysis of previous algorithms, while providing expressions for a comparative analysis of sequential clique enumeration algorithms has also revealed some of the properties desired by a "good" procedure and some of the hazards one must attempt to avoid. In addition, certain properties are characteristic of algorithms for explicitly enumerating the cliques of a graph.

These algorithms appear to require a means of determining the sets of vertices adjacent to a given vertex as all procedures discussed use this information to generate the components to be used in further analysis. This is not too surprising since any graph is characterized by this sort of information. However, the adjacency matrix representation of a graph provides this most simply and directly. The representation of the rows of the adjacency matrix as a string of bits greatly simplifies the computation required in determining new components. The importance of an adjacency matrix representation over some other representation is thus emphasized by these observations.

The desirable properties of a sequential clique enumeration algorithm are two-fold. First, generate new components which do not destroy the existence of maximal complete subgraphs with as little effort as possible. Secondly, generate as few such components as possible. The best possible situation is to avoid the need for determining whether a complete subgraph or component just generated is

properly contained in some other clique of the graph and requires some means of choosing just the right set of vertices so that no redundant component is ever generated. Because of the complexity of the possible intersections of the vertex sets of cliques in a graph, it is difficult to determine just how such a set could be chosen. It is not sufficient to find either a maximal or a minimal independent set of vertices which covers the vertex set of a graph as the following example illustrates. Consider the graph of Fig. 2.9. The vertices labelled 1 and 4 in the graph constitute a minimal independent set of vertices covering the vertex set of the graph. It is clear however that the clique K(1,1,1,1) induced on vertices 2,3,5 and 6 contains no vertex in this particular minimal covering.

Similarly consider the graph in Fig. 2.10. It has a maximal independent set of vertices labelled 1,3 and 5, none of which is a member of the clique induced on vertices 2,4, and 6.

Clearly one requires the prescient ability to choose an appropriate independent covering set of independent vertices among an exponential number of possible choices. There is presently no known way for accomplishing such a task in an efficient manner. The algorithm to be described generates a reduced number of redundant vertex sets, and uses an efficient procedure for detecting such redundancy.

It will be seen that with some modifications the procedure to be described combines some of the better features

of the previous algorithms. As a consequence we shall show that the performance of this algorithm is comparable to and in many cases (eg. Graphs with many cliques) better than that to be expected from the others.

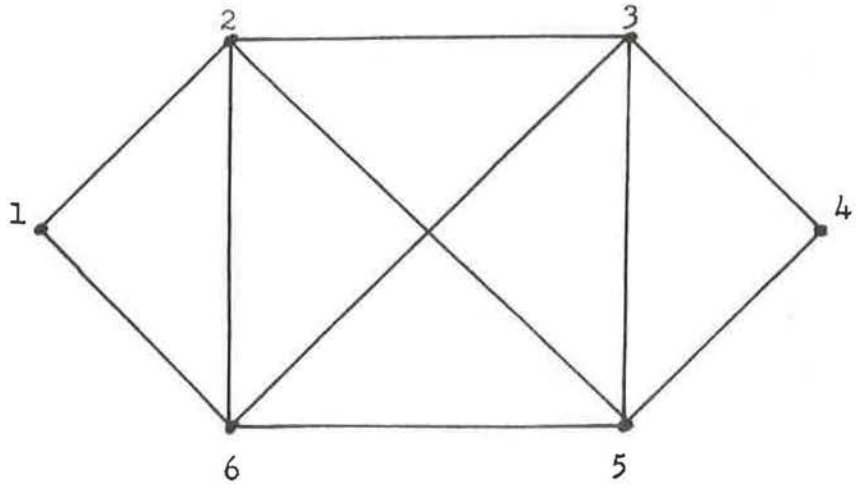


Fig. 2.9

MINIMAL COVER COUNTEREXAMPLE

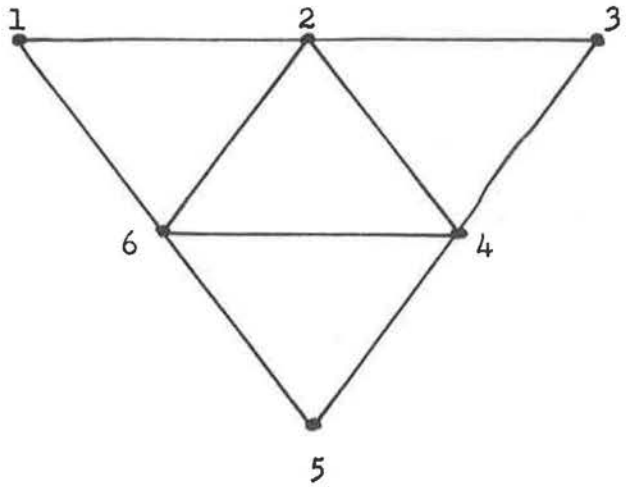


Fig. 2.10

MAXIMAL COVER COUNTEREXAMPLE

2.6.1 DESCRIPTION OF THE ALGORITHM

Each vertex subset to be processed is divided into two parts; V(G1), the set of all vertices in that subset yet to be examined and V(G2), the set of all vertices previously examined and which induce a complete subgraph in the original graph. The vertices in V(G1) have the additional property that they represent all possible extensions of the complete subgraph induced on V(G2) which yield a larger complete subgraph.

If V(G1) is empty, then provided there does not exist a vertex adjacent to all vertices in V(G2), we have found a clique. Such a condition is maintained by deleting from further consideration any vertex subset all of whose members are adjacent to some vertex outside the subset.

If V(G1) is not empty then we generate n - d(v) new sets by first choosing a vertex v, and then considering it together with the n - d(v) - 1 vertices not adjacent to v. Each vertex from this set is used to define a new vertex subset by adding it to V(G2) and thus extending the set of vertices already considered, and then defining a new set of vertices to be considered from V(G1) by including only those vertices adjacent to that vertex just added to V(G2).

2.6.2 NOTATION

V(G1): set of vertices in the current vertex set yet to be considered.

V(G2): set of vertices in the current vertex set which induces a complete subgraph.

V(H1): new set of vertices to be considered.

V(H2): new expanded set of vertices inducing a complete subgraph.

F: set of vertices not adjacent to a chosen vertex from V(G1).

A: the adjacency matrix of the subgraph induced on $V(G1) \cup V(G2)$.

2.6.3 REDUCED REDUNDANCY ALGORITHM

STEP0: initially place $V(G) \cup \phi$ on the stack.

STEP1: Choose a vertex subset $V(G1) \cup V(G2)$ from the stack of subsets to be considered. If stack empty, stop.

STEP2: If there exists a vertex adjacent to all vertices in $V(G1) \cup V(G2)$ then go to STEP1

STEP3: If V(G1) is empty then print V(G2) as a clique and go to STEP1.

STEP4: Choose a vertex v in V(G1) and define F to be a set consisting of v together with all vertices not adjacent to v.

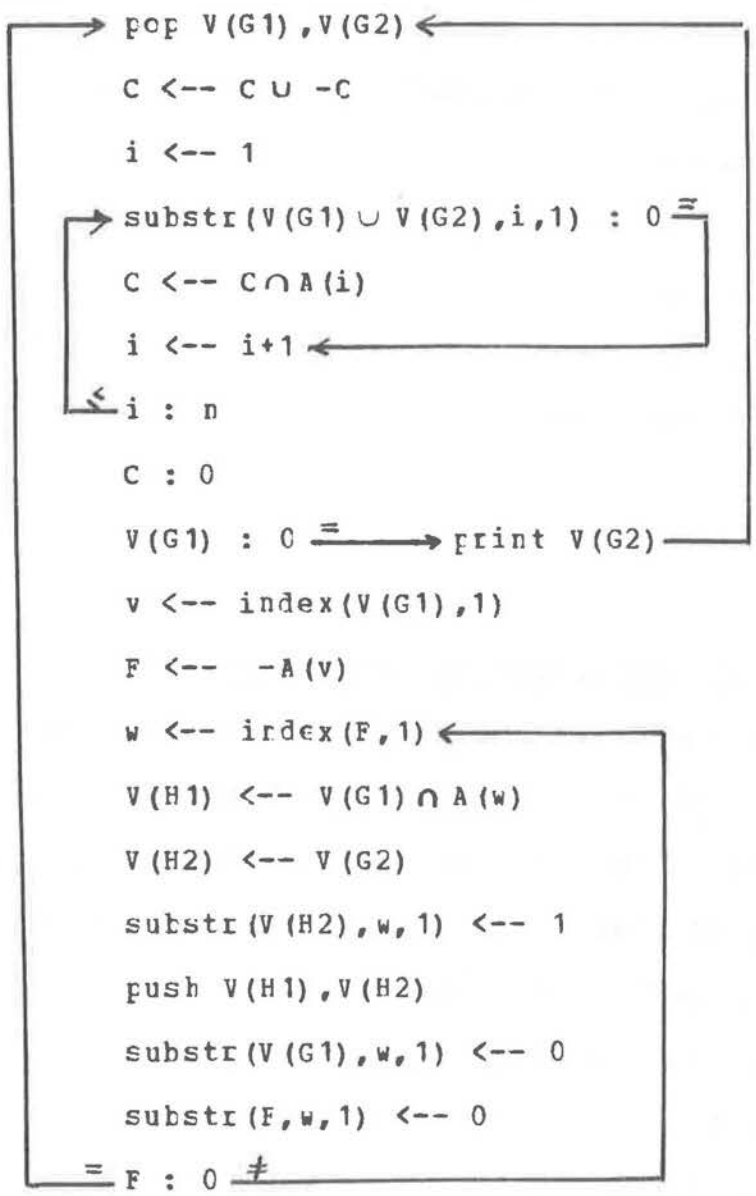
STEP5: Choose a vertex w in F and define a new subset $V(H1) \cup V(H2)$ where V(H1) is the set of all vertices in V(G1) adjacent to w, and $V(H2) = V(G2) \cup \{w\}$.

STEP6: Delete vertex w from sets V(G1) and F.

STEP7: If F empty then go to STEP1; else go to STEP5

In order to compute the time for one iteration of the

algorithm the instructions performed are as follows:



2.6.4 NUMBER OF VERTEX SETS GENERATED

As an example we again consider the derivation tree of K(3,3,3) labelled as previously in Fig. 2.1, the tree this time being determined by our algorithm. It is given in Fig. 2.11. Each vertex not representing a clique is labelled by the

pair (V(G1),V(G2)) representing the vertex subset generated by the algorithm. The number of nodes in the derivation tree for $K(m^k)$ is given in the following theorem:

THEOREM 2.7: The number of nodes in the derivation tree of $K(m^k)$ is $\frac{m^{k+1}-1}{m-1}$.

Proof: Let the nodes of $K(m^k)$ be labelled such that if v_i is a vertex of block V_i, v_j a vertex of block V_j and $i < j$, then the label of v_i is smaller than the label of v_j. The algorithm processes the vertices of a graph in ascending order of labelling. A typical component during execution of the algorithm has i vertices in V(G2), one from each of the blocks $V_1, V_2, \ldots, V_i$. V(G1) induces a complete (k-i) partite graph with m vertices per block. This component therefore determines m new components, one for a selected vertex in block V_{i+1} and m-1 for the m-1 other vertices in V_{i+1} the only vertices not adjacent to the selected one. Each component therefore determines m new ones, until there are k vertices in V(G2) in which case there are none in V(G1). The number of vertices in the derivation tree is therefore $\sum_{i=0}^{k} m^i = \frac{m^{k+1}-1}{m-1}$.

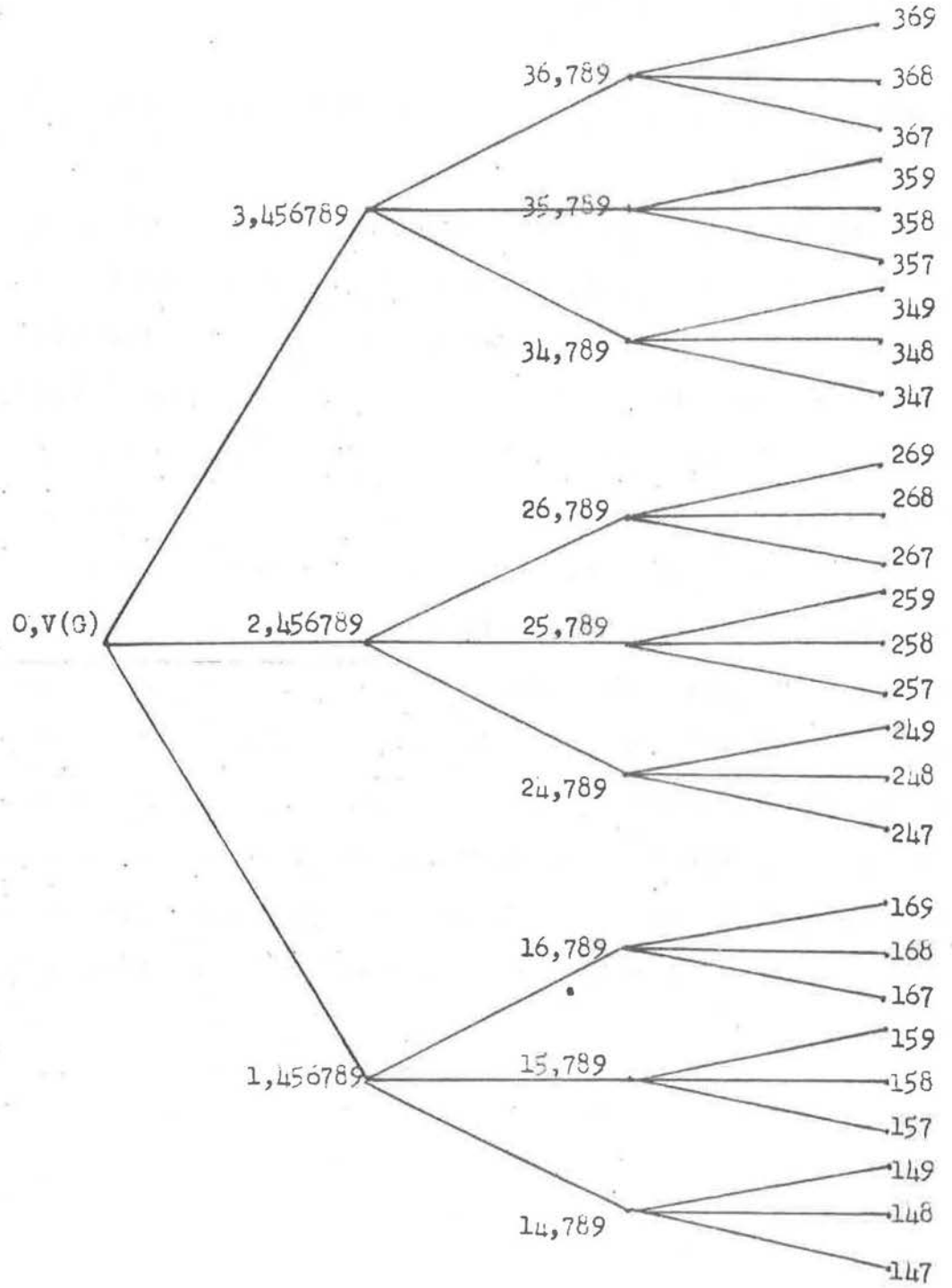


Fig. 2.11

REDUCED REDUNDANCY ALGORITHM: DERIVATION TREE

2.6.5 ANALYSIS OF AN ITERATION

The computational effort for one iteration of the algorithm to find the cliques of $K(m^k)$ is easy to determine from the Iverson description. The loop:7 to 4 is executed n-1 times where n is the number of vertices in the original graph, while the loop: 19 to 12 is determined as follows. Let the vertex set currently under consideration have i vertices in V(G2), and (k-i)m vertices in V(G1). Then F defined in line 11 consists of all vertices in one block and consequently loop: 19 to 12 is executed m-1 times.

The expression for the computation time during one iteration is therefore given by

$$T_0(n) = C_1^0 + nC_2^0 + mC_3^0$$

where

$$C_1^0 = 4t_1 + t_2 2t_4 + t_7 + t_9$$
$$C_2^0 = 2t_1 + t_3 + 2t_6 + t_8$$
$$C_3^0 = 6t_1 + t_2 + t_4 + t_6 + 3t_8 + t_9$$

provided the vertex set under consideration does not have V(G1) empty. If V(G1) is in fact empty, as it will be for all nodes of the derivation tree representing cliques, then only lines 1 through 9 are performed and the computation time in this instance is $T_1(n) = C_1^1 + nC_2^0$ where C_2^0 is given above and $C_1^1 = 2t_1 + t_2 + 2t_4$.

We can now combine the results of the computation time for one iteration with the number of nodes in the derivation tree to obtain an expression for the total computation time required to find the cliques of $K(m^k)$. There are m nodes for

which the computation time for one iteration is $T_1(mk)$, and $\frac{m^{k+1}-1}{m-1} - m^k = \frac{m^k-1}{m-1}$ nodes where $T_0(mk)$ is the computation time. Hence $T(mk) = T_1(mk)m^k + T_0(mk)\frac{m^k-1}{m-1}$, $m>1$. The case for $m = 1$ clearly defines a derivation tree consisting of a single path of length k. Hence the computation time to determine that $K(1^k)$ is a clique is $T(k) = T_1(k) + (k-1)T_0(k)$.

2.6.6 STORAGE REQUIREMENTS

Like the previous algorithms of Harary-Ross, Bonner, and Peay, the Reduced Redundancy algorithm maintains only one adjacency matrix, that of the original graph G. Vertex subsets are maintained on a stack and used to select the appropriate rows of the adjacency matrix of G, to obtain adjacency properties of the subgraph of G induced on the vertices in the vertex subset. Our algorithm, however, generates $n - d(v)$ new vertex sets during an iteration where n is the number of vertices in the set and $d(v)$ is the number of vertices in that set adjacent to v. For this reason it is more difficult to determine the storage requirements of the push-down store for an arbitrary graph. Instead we shall again examine the complete k-partite graph $K(m^k)$.

If we again adopt the strategy of developing the derivation tree in a "depth before breadth" manner, it is clear that no path is of length greater than k. Further, from the previous discussion we know that every vertex set $V(G1) \cup V(G2)$ is complete k-partite with 1 vertex in each of i

blocks. V(G1) consists of the m(k-i) vertices in the remaining k - i blocks. If we select a vertex v from V(G1) to determine new components, the number of vertices not adjacent to v is m-1. Therefore, to each node in a path of length k in the derivation tree there are m-1 other nodes corresponding to vertex sets yet to be processed. Hence the push-down store must be capable of handling k(m-1) vertex sets.

The storage requirements for the new algorithm applied to $K(m^k)$ are $k(m-1)+(mk)^2+2mk+16C$ bits, C being the number of counters and pointers used. Since we can partition a graph into k blocks no block of which has more than m vertices for $k = \chi(G)$, the chromatic number of the graph, this expression also serves as an upper bound on the storage requirements for an arbitrary graph.

2.7 THE BRON-KERBOSCH ALGORITHM

The Bron-Kerbosch algorithm is the most recent clique enumeration algorithm known to this author. Mulligan [64] has described the procedure and found it to be superior to Bierstone's algorithm, a method also discussed by Augustson and Minker [5].

The algorithm employs a recursive procedure which is used to modify a global vertex set consisting of all vertices which form a complete subgraph of the original graph. The function of the recursive procedure is to extend, if possible, the number of vertices in the complete subgraph. This is accomplished by maintaining several lists and pointers in a stack generated through recursive calls to the procedure. These include two vertex subsets, one a set of candidates which can be used to extend the complete subgraph, and the second a set of vertices which have already been used to carry out such an extension. Since the contents of the vertex subset is under continual modification it is also necessary to maintain a pointer indicating the last entry into the set. Some other counters are also maintained through recursive stacking of definitions which will be apparent from the description of the algorithm. In what follows we shall use the notation developed by Mulligan and his formulation of the algorithm.

2.7.1 NOTATION

n: number of vertices in the original graph.

Compsub: complete subgraph currently being extended.

C: order of compsub.

V: vertex set currently under consideration.

Ne: number of vertices already examined in V.

Ce: total number of vertices in V.

S: pointer to selected vertex from V used to extend compsub.

Pos: position of a potential candidate.

Nod: number of vertices not adjacent to a fixed vertex among the set of vertices in V already examined.

Minnod: minimum number of vertices not adjacent to a fixed vertex.

Fixp: vertex with maximum degree in the subgraph induced on V.

NEW: new vertex set.

Newne: number of vertices in NEW that have been examined before.

Newce: total number of vertices in NEW.

2.7.2 THE BRON-KERBOSCH ALGORITHM (Mulligan [64])

A. Initial call to recursive procedure EXTEND

STEP1: Set V to V(G), c to 0 ne to 0, ce to n.

STEP2: Call recursive procedure EXTEND (V,ne,ce). On return stop.

B. EXTEND (V,ne,ce) recursive procedure.

STEP1: set minnod to ce, nod to 0 , i to 0.

STEP2: set i to i+1. If i>ce or minnod = 0 then go to STEP6.

STEP3: set count to 0. For each V(j), j=ne+1 to ce not adjacent to V(i) set count to count+1 and pos to j.

STEP4: if count < minnod then set fixp to V(i), minnod to count and go to STEP5; else go to STEP2.

STEP5: if i ≤ ne then set s to pos; else set s to 2 and nod to 1. In either case go to STEP2.

STEP6 Set nod to minnod+nod.

STEP7: If nod ≤ 0 then return.

STEP8: Interchange V(s) with V(ne+1).

STEP9: Set newne equal to the number of vertices in {V(1),V(2),...,V(ne)} adjacent to V(ne+1). Set NEW(1),NEW(2),... ,NEW(newne) equal to those vertices.

STEP10: Set NEW(newne+1),...,NEW(newce) equal to those vertices in {V(ne+2),...,V(ce)} adjacent to V(ne+1). Newce equals the total number of vertices in NEW.

STEP11: set c to c+1, compsub(c) to V(ne+1).

STEP12 If newce equals 0 then print compsub(i), i = 1,2,...,c as a clique; else if newne less than newce then call EXTEND(NEW,newne,newce)

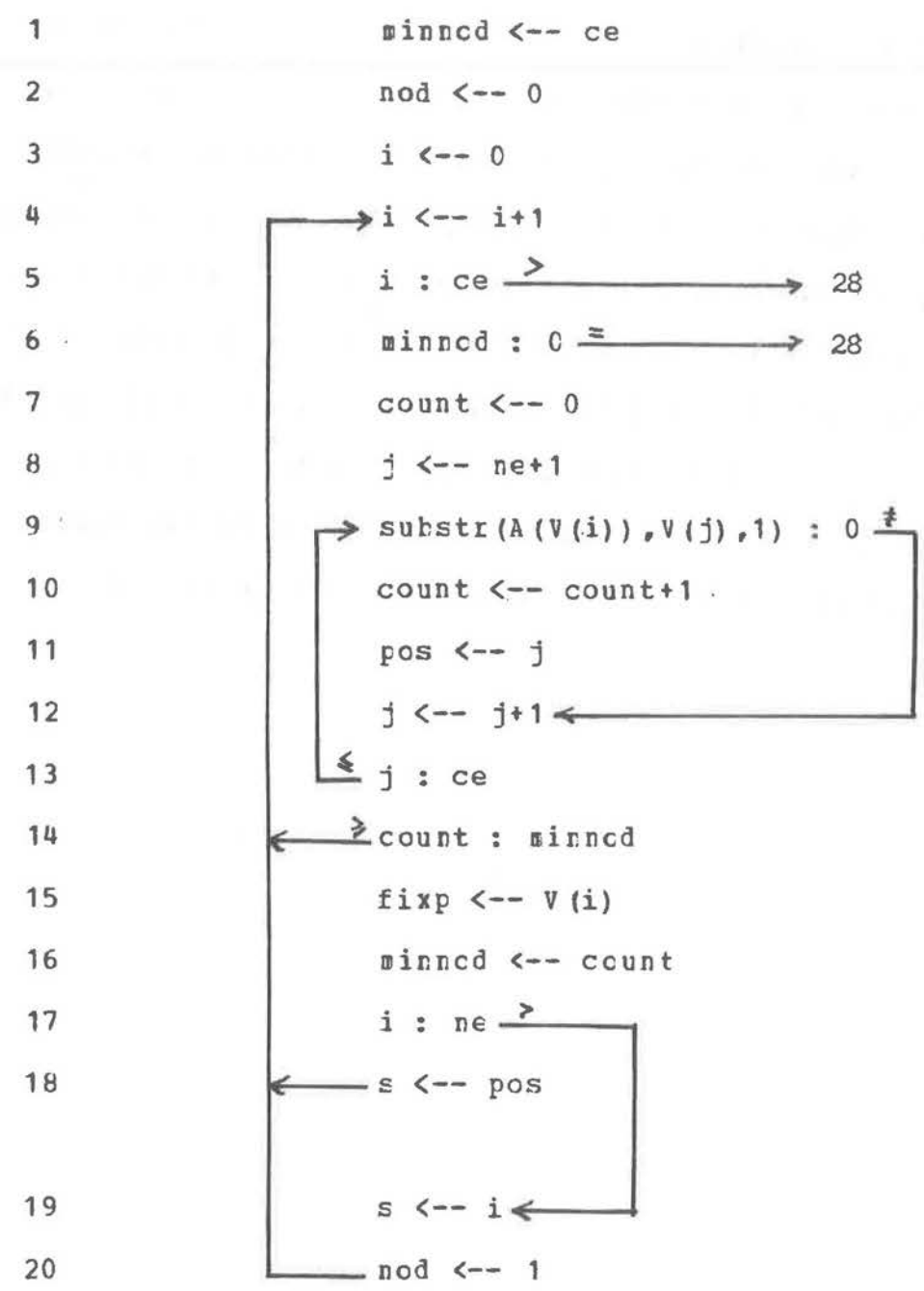
STEP13: Set c to c-1, ne to ne+1.

STEP14: set nod to nod-1. If nod > 0 then choose another vertex from {V(ne+1),...,V(ce)} not adjacent to fixp and not yet chosen. Go to STEP7.

The Bron-Kerbosch algorithm has been divided into three blocks. The task performed by block 1 is to extend if possible

a complete subgraph contained in G by examining vertices not previously encountered to see whether they are adjacent to all of the vertices of the complete subgraph under consideration. Control is passed to block 3 where if such an extension is possible it is made, a record being kept of those vertices previously encountered and yet to be examined. If the complete subgraph cannot be extended it is printed out. Block 3 recursively calls block 1 returning only after all possible extensions have been examined. Control is then passed to block 2 which makes the next possible extension to the vertex set under consideration at the present level of recursion.

BLOCK1


1 minncd <-- ce
2 nod <-- 0
3 i <-- 0
4 i <-- i+1
5 i : ce > 28
6 minncd : 0 = 28
7 count <-- 0
8 j <-- ne+1
9 substr(A(V(i)),V(j),1) : 0 ≠
10 count <-- count+1
11 pos <-- j
12 j <-- j+1
13 j : ce ≤
14 count : minncd ≥
15 fixp <-- V(i)
16 minncd <-- count
17 i : ne >
18 s <-- pos
19 s <-- i
20 nod <-- 1

BLOCK2

```
21          c <-- c-1
22          ne <-- ne+1
23          nod <-- nod-1
24          s <-- ne+1
25      ->  substr(V(s),fixp,1) : 0 ≠--> 29
26          s <-- s+1
27      <=  s : ce
            return
```

BLOCK3

```
28          nod <-- minncd+ncd
29          nod : 0 =--> return
30          sel <-- V(s)
31          V(s) <-- V(ne+1)
32          V(ne+1) <-- sel
33          newne <-- 0
34          i <-- 1
35      ->  substr(A(V(i)),V(ne+1),1) : 0 =
36          newne <-- newne+1
37          NEW(newne) <-- V(i)
38          i <-- i+1 <-------------------
39      <=  i : ne
40          newce <-- newne
41          i <-- ne+2
42          substr(A(V(i)),V(ne+1),1) : 0 =--> 45
43          newce <-- newce+1
```

```
44                  NEW(newce) <-- V(i)
45                  i <-- i+1
46          42 <=   i : ce
47                  c <-- c+1
48                  compsub(c) <-- V(ne+1)
49                  newce : 0 =
50                  i <-- 1
51              ->  print compsub(i)
52                  i <-- i+1
53              <=  i : c
54          21 <=   newne : newce
55          21 <--  call EXTEND(NEW,newne,newce)
```

2.7.3 THE NUMBER OF VERTEX SETS GENERATED

Let $V_1, V_2, \ldots, V_k$ be the blocks of vertices of $K(m^k)$, each containing m mutually non-adjacent vertices. The Bron-Kerbosch algorithm proceeds by fixing a vertex and defining a new vertex subset to be the set of all vertices adjacent to the fixed vertex. This vertex subset is partitioned into two parts to provide information for determining whether a complete subgraph is maximal or has been considered before. At a given level i of the recursion other vertex sets are generated whenever control again returns to that level by choosing among the set of vertices not adjacent to the original fixed vertex. This selection procedure is analogous to the mechanism employed by the Reduced Redundancy algorithm for generating new vertex subsets and as a consequence the number of vertices

in the derivation tree is the same, namely $\frac{m^{k+1}-1}{m-1}$. To see that this is the case it is only necessary to establish an equivalence between the nodes of the derivation tree generated by our algorithm and the nodes in the derivation tree generated by the Bron-Kerbosch algorithm. If we assign level 0 to the node of the derivation tree corresponding to the original vertex set of the graph, then a node at level i in the algorithm corresponds to the vertex subset of a complete (k-i) partite graph while the selection of fixed vertices made in the generation of a path from the root to level i is contained in the array compsub . From previous discussion our algorithm has a node a distance i from the root with two vertex sets V(G1),V(G2). G2 corresponds to a complete subgraph of order i, while V(G1) induces a complete (k-i) partite graph. By choosing that path of length i which results in V(G2) containing the same vertices as compsub, V(G1) is then the same set as the vertex subset generated by the Bron-Kerbosch algorithm.

2.7.4 ANALYSIS OF ONE ITERATION

Since the algorithm employs the same technique for vertex set generation as our algorithm, the relative efficiencies of the two procedures are dependent upon how the properties of the vertex set so generated are exploited during an iteration. This depends on three factors; the way the data is represented, the order of development of the derivation tree and the means by which redundant components are avoided.

To determine the computation time required by the Bron-Kerbosch algorithm as implemented by Mulligan let V be a vertex set under consideration at level i. The maximum depth of recursion by the algorithm is k. At level i we have i+1 calls outstanding of which i have been called by the procedure EXTEND itself. All parameters defined within the procedure are saved, a feature important in the determination of storage requirements.

A vertex set generated by the algorithm in finding the cliques of $K(m^k)$ has the property that at level i all vertices that have been considered lie in blocks $V_1, V_2, \ldots, V_i$ while those yet to be considered lie in $V_{i+1}, V_{i+2}, \ldots, V_k$. There are m(k-i) vertices in the latter collection since every vertex is adjacent to any vertex in V_i from which the selected candidate was chosen. Hence every vertex in the last (k-i) blocks of $K(m^k)$ has the minimum number of disconnections since the vertex set under consideration can have at most m-1 vertices from any previous block already considered, the remaining vertex currently in compsub. Since minnod > 0 for all vertices in the vertex subset, loop:14 to 4 is repeated ce-ne-1 times plus once more when choosing a vertex for fixp, while loop:18-to 4 is repeated at most ne times. Finally, the inner loop:13 to 9 is repeated m(k-i) times for each vertex already considered and m(k-i-1) times for each vertex yet to be considered.

The time for one iteration of block 1 is thus bounded by

$$T_1 = C_1^1 + (ce-1)C_2^1 + ne(C_3^1 + m(k-i)C_4^1) + (ce-ne)m(k-i-1)C_4^1$$

where

$$C_1^1 = 7t_1+t_4$$
$$C_2^1 = 3t_1+2t_3+4t_4$$
$$C_3^1 = 3t_1+t_4$$
$$C_4^1 = 3t_1+2t_3+t_4+t_8$$

Loop:54 to 21 is repeated after every return from the recursive call in line 55. This equals n-d(fixp), the number of vertices not adjacent to fixp. Loops 27 to 25 and 46 to 42 are repeated ce-ne times, while loop:39 to 35 is repeated ne times. If i<k-1 a clique has not yet been found so statements 50 through 53 are skipped. If we define one iteration as being the total computation perfcrmed until a return is made at line 29, then the computation time for blocks 2 and 3 together is bounded by

$$T_2 = (n-d(fixp))(C_1^2+(ce-ne)C_2^2+neC_3^2)+t_1+t_3$$

with

$$C_1^2 = 13t_1+6t_3$$
$$C_2^2 = (2t_1+t_3+t_4+t_8)+(t_1+t_3+2t_4+t_8)$$
$$C_3^2 = 3t_1+2t_3+2t_4+t_8$$

The order of the computation time for one iteration is therefore between n and n^2. For vertex subsets such that ne=0, the computation for one iteration is of order $m^2(k-i)^2$, the square of the number of vertices in the subset under consideration.

2.7.5 STORAGE REQUIREMENTS

As previously observed, the maximum depth of recursion is k. Hence all variables defined within the recursive function must be stored k times. This consists of pointers and counters and the array NEW an integer array of size n. The adjacency matrix of the original graph and the array compsub of order n are maintained outside the recursive procedure. Since the adjacency matrix is stored as an array of bit strings the storage requirements for the Bron-Kerbosch algorithm as implemented by Mulligan are $n^2+n(k+1)+16Ck+16$ bits where C is the number of integer scalars defined within the routine EXTEND, and the additional 16 bits are an allowance for a globally defined variable.

CHAPTER 3: CLIQUE DETECTION USING VERTEX SIMILARITY

3.1 INTRODUCTION

It is evident from the observations and results of Chapter 2 that the efficient detection of cliques is severely hampered by the possibly exponential number of such subgraphs. Even the act of printing them out can occupy an inordinate amount of time unless there exists some means of simultaneously identifying several cliques and also some more compact form of notation than explicitly defining the vertex sets of each clique. Two approaches to this problem will be examined separately in this chapter, each of them exploiting properties which measure the degree to which any two vertices are different.

One such device is similarity of vertices. The automorphism group of a graph partitions the set of vertices V(G) into equivalence classes called the orbits of $\Gamma(G)$. Two vertices are similar if and only if they are members of the same orbit. Hence there exists a permutation in $\Gamma(G)$ which maps u onto w where u and w are vertices in the same orbit.

An examination of complete k-partite graphs with m vertices in block v , reveals that every vertex in any block can be interchanged with any other, ie vertices in any block are similar. Let u, w be two such vertices. Then if we know the cliques to which u belongs and a permutation which maps u onto w, we also have all the cliques to which w belongs. Such a representation is more compact as it requires

explicitly defining only the cliques associated with u.

In the development which follows we shall tacitly assume that a procedure for determining the orbits is available. For implementation we shall employ Corneil's algorithm[12]. It is important to note here that while Corneil's procedures have not failed on any graphs encountered to date, that their correctness depends on an unproved conjecture. Corneil therefore describes his algorithm as a heuristic one, a policy which we formally must also follow when using his routines.

A difficulty with such an overall approach as offered here for improving the efficiency of clique detection occurs when u and w, vertices in the same orbit, are both members of some common clique, in other words u is adjacent to w. This is often the case as is illustrated by the existence of connected point symmetric graphs, which by definition have all vertices belonging to the same orbit. Since vertex similarity can be used at several levels of clique detection other than enumeration we shall defer further discussion on this problem until later.

3.2 POINT AND LINE SYMMETRIC GRAPHS

The remarks of the introduction to this chapter suggest that similarity of vertices may contribute to a characterization of the cliques in a graph. To what extent this is true will be examined in this and the next section. Of particular interest will be the behavior of subgraphs induced by a single vertex since this is the major mechanism by which a graph can be decomposed into smaller subgraphs for further processing. This feature has already been observed previously in the sequential algorithms of chapter 2.

The major portion of this section is devoted to an examination of cliques in point-symmetric and line-symmetric graphs. Of particular interest is the degree of symmetry of the induced subgraphs.

THEOREM 3.1: The subgraph induced on the adjacency set of a fixed vertex in a line-symmetric graph is point-symmetric.

Proof: Let G be a line-symmetric graph, v a vertex in V(G), and denote by $\{w_1, w_2, \ldots, w_k\}$ the set of vertices adjacent to v. Since G is line-symmetric there exist permutations $\alpha_2, \alpha_3, \ldots, \alpha_k$ in $\Gamma(G)$ such that:

$$\alpha_2(v, w_1) = (v, w_2)$$
$$\alpha_3(v, w_1) = (v, w_3)$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$\alpha_k(v, w_1) = (v, w_k)$$

These permutations, together with their inverses, hold v fixed and hence belong to Γ_v, the stabilizer of v. Since every α in

$\Gamma(G)$ preserves adjacencies, every α maps w_i cntc some w_j where w_i, w_j, are in $\{w_1,\ldots,w_k\}$. If w_i, w_j are any two vertices in $\{w_1,\ldots,w_k\}$ then $\alpha_j \alpha_i^{-1}(v,w_i) = (v,w_j)$ and hence the set of vertices $\{w_1,\ldots,w_k\}$ is similar. Further, since every permutation $\alpha \in \Gamma_v$ maps w_i onto some w_j the integrity of the subgraph induced on $\{w_1,\ldots,w_k\}$ is preserved.

The subgraph induced on the adjacency set of a fixed vertex in a point-symmetric graph is not necessarily point-symmetric. This is illustrated in the counter-example given in Fig. 3.1. The subgraph induced on vertices adjacent to vertex 1 is given in Fig. 3.2 and is clearly not point-symmetric.

Dauber and Harary [43] and Folkman [32] have investigated the extent to which line-symmetric graphs are point-symmetric. The principal result of Dauber and Harary is the establishment of conditions which characterize such graphs, namely that every line-symmetric graph with no isolated points is point-symmetric or bipartite. This result together with the previous theorem establishes a sufficient condition for point-symmetric graphs to have point-symmetric subgraphs induced on the adjacency set of a fixed vertex, that condition being that the graph be line-symmetric or regular bipartite. That this condition is not necessary is illustrated by the graph given in Fig. 3.3, a graph not line-symmetric or regular bipartite but point-symmetric and every subgraph induced on a set of vertices adjacent to a single vertex is also point-symmetric. For this graph the edge (1,2) is not similar to the edge (1,6).

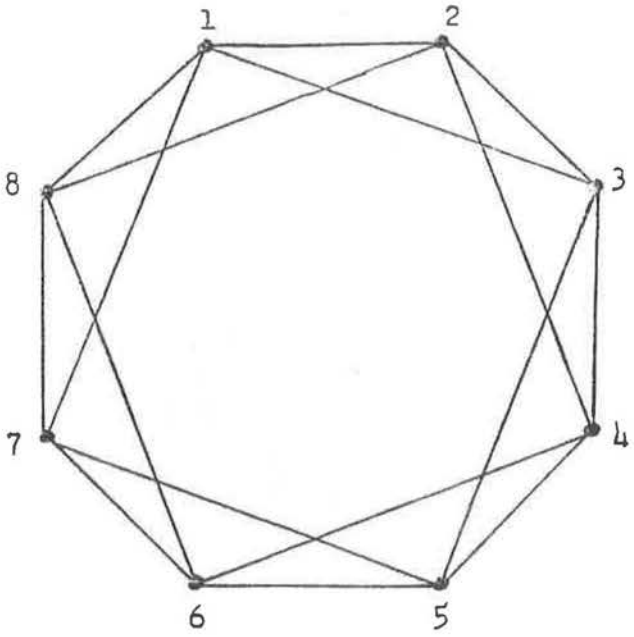


Fig. 3.1

POINT SYMMETRIC GRAPH

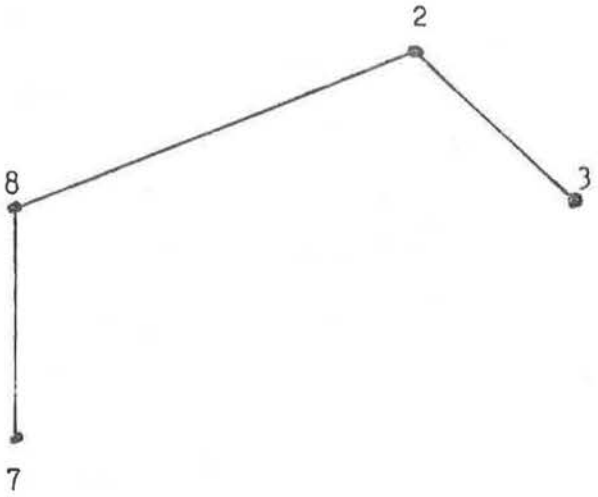


Fig. 3.2

INDUCED SUBGRAPH OF FIG. 3.1

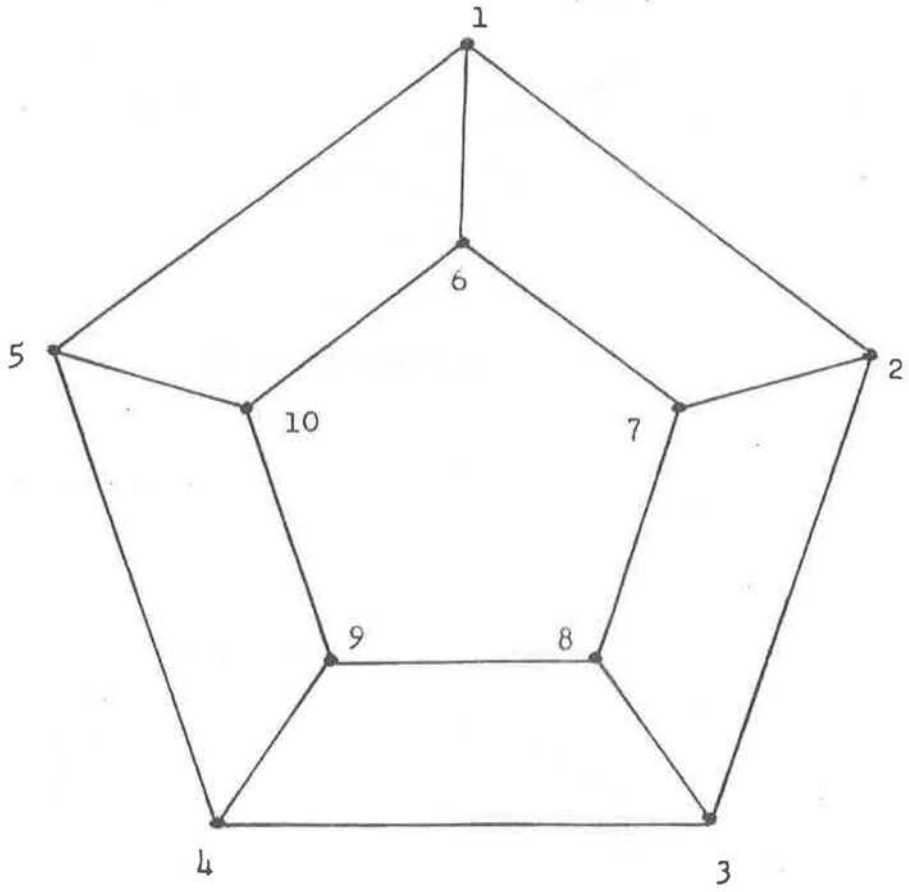


Fig. 3.3

POINT-SYMMETRIC GRAPH NOT LINE-SYMMETRIC

We shall denote by A(v) the set of vertices in a graph G adjacent to the vertex v.

LEMMA 3.1: let v_1, v_2 be any two similar vertices of a graph G, and denote by G_1 and G_2 the subgraphs induced on $A(v_1)$ and $A(v_2)$ respectively. Then there exists α in $\Gamma(G)$ such that $\alpha G_1 = G_2$.

Proof: Since v_1, v_2 are similar the number of vertices in G_1 is equal to the number in G_2. Let α be an automorphism of G mapping v_1 onto v_2. Then for each u in $V(G_1)$ there exists a unique image αu. Further every such vertex αu in $\alpha V(G_1)$ is adjacent to v_2 since every vertex u in $V(G_1)$ is adjacent to v_1. Now by definition $V(G_2)$ is the set of vertices adjacent to v_2, and hence $V(G_2) = V(G_1)$.

Since α is an automorphism, (u,w) is in E(G) if and only if $(\alpha u, \alpha w)$ is in E(G). Hence for any u,w in $V(G_1)$, (u,w) is in $E(G_1)$ if and only if $(\alpha u, \alpha w)$ is in $E(G_2)$ since αu, αw are vertices from $V(G_2)$. Therefore $E(G_2) = \alpha E(G_1)$ and hence $\alpha G_1 = G_2$.

As a consequence of this Lemma we have the following:

LEMMA 3.2: Let G be a point symmetric graph, and denote by $\alpha_2, \alpha_3, \ldots, \alpha_k$ automorphisms of G such that

$$\alpha_2 v_1 = v_2,$$
$$\alpha_3 v_1 = v_3,$$
$$\ldots\ldots\ldots\ldots$$
$$\alpha_k v_1 = v_k.$$

Then it is the case that:

$$\alpha_2 G_1 = G_2,$$
$$\alpha_3 G_1 = G_3,$$
$$\cdots\cdots\cdots\cdots$$
$$\alpha_k G_1 = G_k,$$

where $G_1, G_2, \ldots, G_k$ denote the subgraphs induced on $A(v_1), A(v_2), \ldots, A(v_k)$ respectively.

Let $\{v_1^i, v_2^i, \ldots, v_k^i\}$ be the vertex set of G for $i = 1,2,\ldots,k$ and suppose without loss of generality, that $\alpha_i v_j^1 = v_j^i$. Then clearly if we know the cliques of G_1, we can find all the remaining cliques of G knowing the permutations $\alpha_2, \alpha_3, \ldots, \alpha_k$. To avoid duplication of cliques the following test can be employed. If we are examining the cliques associated with component G_i, then delete all cliques containing vertex v_j for $j = 1,2,\ldots,i-1$ as such cliques have already been found during the examination of component G_j.

Such a strategy encounters difficulties on two fronts. First, as we have seen previously, the point-symmetry of a graph is no guarantee for the point- symmetry of subgraphs induced on vertices adjacent to a point, hence the problem of determining the cliques of G is as yet unresolved. Secondly, the determination of automorphisms $\alpha_2, \alpha_3, \ldots, \alpha_k$ is in general a difficult problem.

We can overcome the first difficulty by generalizing the procedure to include graphs which are not necessarily point-symmetric. Then, the existence of an algorithm for determining

the orbits can be exploited in the following way.

Every member of each orbit can be represented by a single vertex which determines the induced subgraph for further processing. If $\theta = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ is an orbit of $\Gamma(G)$ for some graph G and if $\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_k}$ are permutations in $\Gamma(G)$ such that:

$$\alpha_{i_2} v_{i_1} = v_{i_2}$$
$$\alpha_{i_3} v_{i_1} = v_{i_3}$$
$$\ldots\ldots\ldots\ldots$$
$$\alpha_{i_k} v_{i_1} = v_{i_k}$$

then by an argument simliar to that previously given the cliques associated with $v_{i_2}, \ldots, v_{i_k}$ can be determined if we know the cliques associated with v_{i_1}. The development of an algorithm utilizing such techniques will be the focus of the next section. The object will not be to find all the cliques because of the difficulties associated with determining the permutations which map similar vertices onto each other. Rather, the algorithm shall attempt to find a set of non-similar cliques of a graph G which together with a knowledge of the automorphism group $\Gamma(G)$ will be sufficient to determine all the cliques of the graph. The algorithm can thus be considered as a sub-program which when incorporated with a sub-program for determining the automorphism group will provide a mechanism for finding the cliques.

3.3 DETERMINATION OF NON-SIMILAR CLIQUES

The purpose of the procedure to be described here is to somehow characterize the non-similar cliques of a graph. Two cliques C_1 and C_2 of a graph G are said to be similar if there exists an automorphism α of G such that $\alpha C_1 = C_2$.

A naive approach to the problem of determining the non-similar cliques of a graph which serves to illustrate what we are attempting to find involves generating the "equivalence classes" induced on the set of all unordered k-tuples of vertices of G by the automorphism group $\Gamma(G)$, for k=1,2,....n-1. For any given k, we examine the cliques that are members of each equivalence class, choosing one as a representative member. Since the automorphism group preserves adjacencies, two k-tuples are members of the same class only if the subgraphs of G defined on the vertices represented by the k-tuples are isomorphic. Such a procedure, therefore, clearly provides more information than we desire as we are intersted only in those classes whose k-tuples are the vertices of maximal complete subgraphs.

The mechanism to be employed will depend primarily on the observations of the previous section; namely, that it is possible to generate all the non-similar cliques of a graph by reducing a graph to components equal in number to the orbits of the automorphism group of the graph, each component being the subgraph induced by a vertex from an orbit. Each component will then serve as input and subsets of vertices will thus be generated in a recursive manner analogous to the sequential

procedure described in chapter 2.

There is not, unfortunately, sufficient information to determine all the non-similar cliques of a graph from the orbital partition of V(G) alone. This is illustrated by the following example. Suppose we apply the procedure of choosing vertices as just described to the graph of Fig. 3.3. Since this graph is point-symmetric, one vertex should be sufficient to characterize the "first" vertex of all the cliques. Let that be vertex 1. The subgraph induced on vertices adjacent to 1 consists of three independent vertices 2,5 and 6 each of which belongs to the same block of the orbital partition of V(G). The "second" vertex of all cliques should therefore be characterized by one vertex, say 2. As (1,2) is a clique of the graph and since we have argued that a single vertex and the subgraph induced on adjacent vertices should be sufficient to characterize all cliques of the graph, we would have to claim that all cliques were similar to (1,2) which we know to be false since the graph being examined is not line-symmetric. In fact we have previously observed that edge (1,6) was not similar to edge (1,2) and therefore should also have been generated in the determination of the non-similar cliques of the graph.

We can resolve the two non-similar cliques of the graph of Fig. 3.3 by using not only the orbital partition of V(G) but also by determining the orbits induced on 2,5,6 by the stabilizer of 1. This results in a partitioning of {2,5,6} into two sets {2,5} and {6}. By selecting a representative

vertex from each of these sets we can obtain two non-similar cliques say (12) and (16). This example illustrates the fact that although the group $\Gamma(G)$ of a graph may be transitive on the vertex set $V(G)$ it is not necessarily transitive on $A(v)$. We have previously shown that if G is line-symmetric then this will be the case. Before we present a description of an algorithm for finding the non-similar cliques of a graph from its orbital structure we examine further the orbits of the stabilizer of a particular vertex v in the following two theorems.

THEOREM 3.2: Every automorphism α in the stabilizer, Γ_v, of v is an automorphism of G_v, the subgraph of G induced on the set of vertices adjacent to v.

Proof: Let P be a permutation matrix corresponding to the automorphism α in Γ_v. Without loss of generality assume the vertices of of G are labelled $1,2,\ldots,m$, and the remaining vertices in G are labelled $m+1,m+2,\ldots,n$. Let $A(G)$ be the adjacency matrix of G, $A(G_v)$ that of G_v. Clearly $A(G)$ is of the form:

$$\begin{bmatrix} A(G_v) & A_2 \\ A_2^T & A_3 \end{bmatrix}$$

Since P corresponds to an automorphism in the stabilizer of v, by Lemma 3.1 P maps $V(G_v)$ onto $V(G_v)$ and is therefore of the form:

$$\begin{bmatrix} P_1 & 0 \\ 0 & P_2 \end{bmatrix}$$ where P_1 acts on vertices labelled $1,2,\ldots,m$, the vertices in $V(G_v)$.

Since is an automorphism of G, it is the case that $P^T A(G) P = A(G)$. Hence

$$\begin{bmatrix} P_1^T & 0 \\ 0 & P_2^T \end{bmatrix} \cdot \begin{bmatrix} A(G_v) & A_2 \\ A_2^T & A_3 \end{bmatrix} \cdot \begin{bmatrix} P_1 & 0 \\ 0 & P_2 \end{bmatrix} = \begin{bmatrix} A(G_v) & A_2 \\ A_2^T & A_3 \end{bmatrix}$$

or $P_1^T A(G_v) P_1 = A(G_v)$. Consequently α maps G_v cntc itself. QED.

As a corollary to this theorem we have following result:

COROLLARY: Every orbit of Γ_v is contained in some orbit of $\Gamma(G_v)$.

Proof: By the previous theorem every automorphism of G is one of G . Let u,w be vertices in $V(G_v)$ and suppose there exists α in Γ_v such that $\alpha u = w$. Hence α is an automorphism of G_v and u and w must be members of the same orbit induced on $V(G_v)$ by $\Gamma(G_v)$. QED.

THEOREM 3.3: Let G be a connected point-symmetric graph. Then G is line-symmetric if and only if for an arbitrary vertex v, the stabilizer $\Gamma_v \subseteq \Gamma(G)$ of v is transitive on A(v) the set of vertices adjacent to v.

Proof: Let $A(v) = \{w_1, w_2, \ldots, w_k\}$. If G is line-symmetric then for any $w_i, w_j, i \neq j$, (v, w_i) is similar to (v, w_j). Hence α is transitive on A(v).

Conversely suppose Γ_v is transitive on A(v). Then for any w_i, w_j there exists α in Γ_v such that $\alpha(v, w_i) = (v, w_j)$. Further since G is point-symmetric, for any other vertex $u \neq v$ in V(G), there exists β in $\Gamma(G)$ such that $\beta v = u$. If $A(u) = \{x_1, x_2, \ldots, x_k\}$ then since preserves adjacencies

$A(u) = \{\beta w_1, \beta w_2, \ldots, \beta w_k\}$ and hence $\beta(v, w_i) = (u, x_j)$ for some x_j in $A(u)$. Finally every edge $(u, \beta w_i)$ is mapped onto $(u, \beta w_j)$ by the automorphism $\beta\alpha\beta^{-1}$, hence every edge incident to a vertex u or v is similar to any other edge also incident to a vertex u or v. Since u was chosen arbitrarily we can conclude that G is line-symmetric.

The observations made in the previous theorems provide the machinery by which we can define an algorithm for determining the non-similar cliques of a graph provided we are equipped with a procedure for determining orbital partitions.

3.4 DESCRIPTION OF THE ALGORITHM

Since two vertices which are members of the same orbit will be members of similar cliques, we initially determine k representative vertices one for each of the k orbits of the graph. Further, we shall require a knowledge of the orbits of the respective stabilizers of the representative vertices.

The algorithm recursively decomposes subgraphs defined on the set of vertices adjacent to a representative vertex into as many new subgraphs as the number of blocks of the orbital partition of the stability subgroup fixing that particular representative vertex. Each new subgraph is determined as the subgraph induced on the set of vertices of the old subgraph adjacent to a single vertex chosen from one of the blocks of the orbital partition and is subsequently reduced in a similar manner.

A record is maintained of the representative vertices as they are chosen, and when there exists an isolated vertex in a block of the partition, a complete subgraph has been found. This subgraph is then examined to see if it is maximal by a procedure similar to that of the Reduced Redundancy algorithm in the previous chapter.

As stated previously, to determine the orbits we shall employ Corneil's algorithm for constructing the Terminal Quotient Graph, a graph each of whose vertices, it is conjectured, corresponds to a block of the orbital partition of V(G) [13]. Corneil's algorithm is ideally suited to our purposes since in determining the Terminal Quotient Graph, he determines not only the vertices of the original graph belonging to each block of the partition but also the orbits of Γ_v for a vertex v from each orbit of $\Gamma(G)$. Since the adjacency set of v is obviously a subset of V(G) it is easy to determine the orbits of Γ_v to which they belong. Corneil's algorithm provides this information in the determination of the vertex quotient graphs of G which are constructed by fixing a vertex and then determining the partition induced on the remaining vertices of V(G). Corneil uses the vertex quotient graphs to determine the orbits of $\Gamma(G)$ by grouping two vertices in the same class if and only if they have identical vertex quotient graphs.

NOTATION

θ_i: the ith orbit of $\Gamma(G)$.

A(v): row v of the adjacency matrix of G.

G1: vertices yet to be considered for a particular subgraph.

G2: vertices which induce a complete subgraph.

H1: new vertex set to be examined, derived from G1.

H2: expanded vertex set inducing a complete subgraph.

NON-SIMILAR CLIQUES ALGORITHM

STEP1: Use Corneil's algorithm to find the orbits $\theta_1^\circ, \theta_2^\circ, \ldots, \theta_k^\circ$ of $\Gamma(G)$. In addition, let $\theta_1^v, \theta_2^v, \ldots, \theta_{k_v}^v$ be the orbits of Γ_v induced on A(v).

STEP2: Choose a vertex set (G1,G2,w) from the stack of candidates. If stack empty, then stop.

STEP3: Compute $T = \overbrace{v \in G1 \cup G2}\ A(v)$. If T not empty then go to STEP2.

STEP4: If G1 empty then print G2 as a clique and go tc STEP2.

STEP5: Set i to 1 and F to G1.

STEP6: If $\theta_i^w \cap G1$ empty then go to STEP10.

STEP7: Choose v in $\theta_i^w \cap G1$ not previously chosen. If none left to examine, let v be any vertex in $\theta_i^w \cap G1$ and gc tc STEP9.

STEP8: If $G2 \cap A(v)$ not empty then go to STEP9; else go to STEP7.

STEP9: Define a new vertex set (H1,H2,v) with $H1 = F \cap A(v)$ and $H2 = G2 \cup \{v\}$. Put (H1,H2,v) on the stack.

STEP10: set i tc i+1 and F to $F \cap (\sim\theta_i^w)$. If $i \leq k$ then go to

STEP6; else go to STEP2.

3.5 DISCUSSION OF THE ALGORITHM

The algorithm for enumerating non-similar cliques of a graph is similar to the sequential algorithm for the enumeration of all cliques proposed in Chapter 2. However, whereas the sequential algorithm's efficiency was dependent upon the number of cliques in the graph and the number of elements in a vertex subset, the determination of non-similar cliques by the method just described is dependent upon the similarity of vertices in the graph. It is clear that this determines the number of orbits of the group as well as the number of non-similar cliques. Since it is only necessary to consider one vertex from each of the blocks of the partition of A(v) induced by the stabilizer of an appropriate vertex v, the number of vertices that need be examined and hence the number of new vertex subsets generated is reduced if the number of blocks in the orbital partitions determined in step1 is small.

It is possible for a graph to have an exponential number of cliques, none of which is similar to any other. This is illustrated by the graph of Fig. 3.4. The subgraph induced on vertices {1,2,...,8} is K(3,3,2) the graph on eight vertices with maximum number of cliques. Additional vertices are then added to insure that the graph has identity group. Hence every clique of G is non-similar to every other. In general it is

possible to construct a graph on 6k vertices having order 3^k cliques in a similar manner. The purpose of this demonstration is to emphasize the fact that the detection of non-similar cliques may itself be an exponential process.

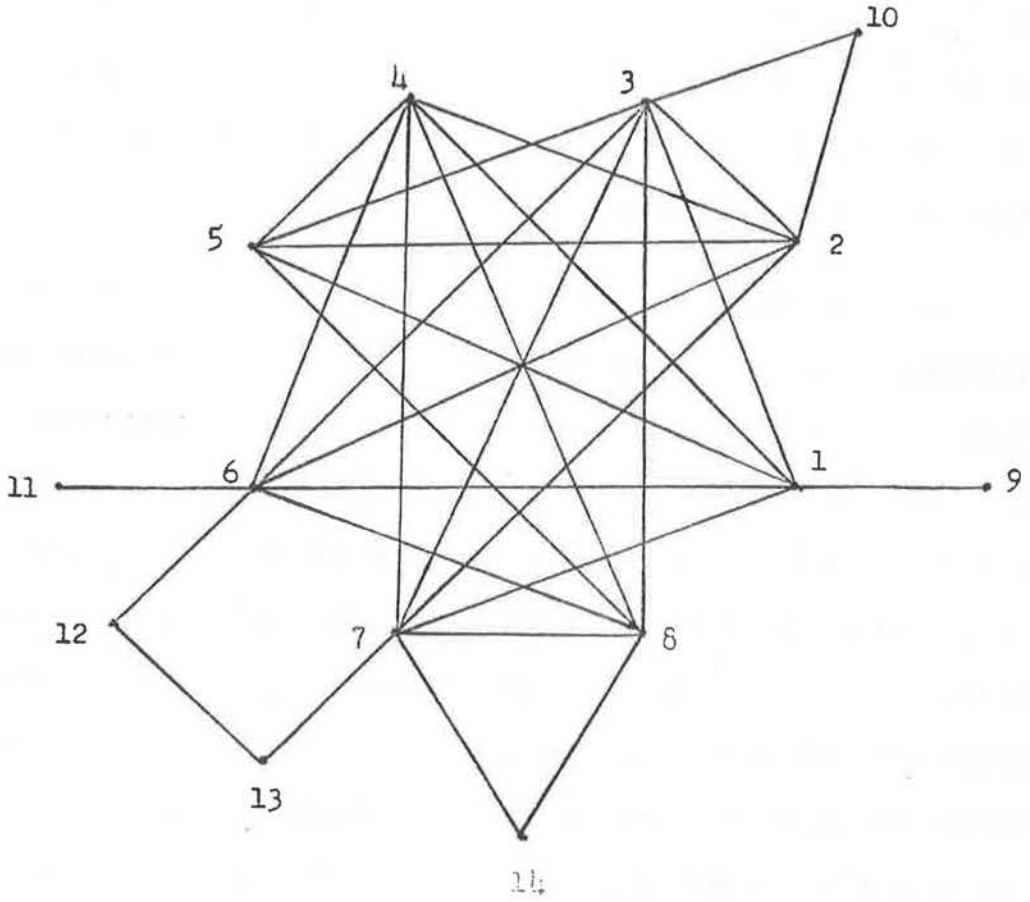


Fig. 3.4

GRAPH WITH ALL CLIQUES NON-SIMILAR

3.6 ANOTHER APPLICATION OF ORBITAL PARTITIONING

In the algorithm for determining the non-similar cliques of a graph, it was not possible for us to apply orbital partitioning to the vertex sets of each component obtained in the reduction of the graphs. This was because the non-similar cliques were determined by the orbits of Γ_v of v, and not $\Gamma(G_v)$. We have previously shown that for every automorphism of Γ_v, there is an automorphism of $\Gamma(G_v)$. However the converse is not necessarily true since two similar vertices in G might be non-similar in G_v. If grouped in the same class, a non-similar clique would be lost.

We can however employ this strategy if we wish to determine only the existence of cliques of different orders. Such a technique is seen to examine fewer vertex subsets than a procedure for finding the non-similar cliques of a graph, since we can take advantage of any symmetry that exists in the subgraph induced on a particular vertex subset. The vertex sets which are nearly resolved into cliques exhibit a high degree of vertex similarity and by only distinguishing between vertices in different orbits, the number of vertices examined is greatly reduced. The fact that cliques of all orders originally present in the graph will be obtained is established by the following argument.

If we determine the orbits of $\Gamma(G)$ on V(G), two vertices u,v in V(G) are members of the same orbit if and only if the subgraphs induced on those vertices of V(G) adjacent to u and those adjacent to v are isomorphic. Hence each induced

subgraph has the same number of cliques of each order and consequently either induced subgraph may be chosen for further processing and the other ignored without fear of losing all cliques of a particular order.

THE ALGORITHM

STEP0: Initially place (G1, ϕ) on the stack.

STEP1: Choose a vertex set(G1,G2) from the stack. If the stack is empty ,then stop.

STEP2: Compute T = $\bigcap_{v \in G1 \cup G2} A(v)$. If T is not empty then go to STEP1.

STEP3: If G1 is empty then print G2 and go to STEP1.

STEP4: Determine the orbits $\theta_1, \theta_2, \ldots, \theta_k$ of the automorphism group of the subgraph induced on vertex set G1.

STEP5: Set i to 1 and F to G1.

STEP6: Choose v in $\theta_i \cap G1 \cap (A(v))$ and define a new vertex set (H1,H2) where H1 = $F \cap A(v)$ and H2 = $G2 \cup \{v\}$. Place (H1,H2) on the stack.

STEP7: Set i to i+1 and F to $F \cap (\sim\theta_i)$. If $i \leq k$ then go to STEP6; else go to STEP1.

This algorithm is very similar to the Reduced Redundancy algorithm for the enumeration of cliques. It is obvious that the latter algorithm could be employed to determine the orders of the non-isomorphic cliques of a graph. However in view of their possibly exponential number, it is desirable to find some means of reducing the number of vertex subsets generated by reducing the number of vertices that need to be examined.

This is achieved in our algorithm by again exploiting the fact that two similar vertices belong to the same number of cliques of different orders and therefore in the situation where we wish to find the orders of the different sized cliques of the graph, it is only necessary to treat one of the two similar vertices.

It should be noted that in step 6 of the algorithm it is not sufficient to choose one vertex from each of the k orbits of the automorphism group of the subgraph induced on G1. This is because it may turn out that the number of orbits exceeds the number of vertices not adjacent tc v, in which case our algorithm would perform more poorly during that iteration than the Reduced Redundancy algcrithm of the previous chapter since it would generate more new vertex sets than the sequential procedure. For this reason v is chcsen frcm $\theta_i \cap G1 \cap (\sim A(v))$.

3.7 ANOTHER APPROACH TO DESCRIBING THE CLIQUES

In the previous sections we have attempted to explcit the similarity of vertices in a graph as an aid to the detection of its cliques. This was only partially successful, cne of the major difficulties being the difficulty of determining the group of the graph, which was necessary for a complete enumeration of the cliques. Even the task of determining the non-similar cliques has proved to be limited by the existence of few good procedures for finding the orbital partition of the vertex set. Finally, we saw where it was even possible for a graph with identity group to have an exponential number of non-similar cliques.

In this sectien we shall explore an alternative approach in which two vertices will be related by a condition stronger than that of similarity.

DEFINITION 3.1: Twc vertices u and w of a graph G are said to be complete subgraph equivalent (CS equivalent) if for any subgraph of G defined on vertices $u,v_1,v_2,\ldots,v_j$ cf V(G), $u,v_1,v_2,\ldots,v_j$ are mutually adjacent if and only if $w,v_1,v_2,\ldots,v_j$ are mutually adjacent. Two vertices of degree 0 are CS equivalent.

It is clear from the definition that if two vertices are CS equivalent then they are similar. This fcllcws from the fact that two CS equivalent vertices are adjacent to the same set of vertices and can be interchanged. By finding all the cliques to which vertex u belongs, we have also found all the

cliques to which vertex w belongs and it is a simple matter to determine the latter explicitly: for each occurrence of u in a clique, replace it by w.

A supplementary but equally important advantage of a method which reduces the number of vertex subsets to be considered by finding complete subgraph equivalent vertices is that it provides a means of representing all the cliques of the graph in a more concise manner than explicit enumeration. Given the vertex set V(G) of a graph G, let $v_1, v_2, \ldots, v_k$ be a set of CS equivalent vertices, all of which are by definition adjacent only to vertices in $A(v_1)$, the set of vertices adjacent to v . If we denote by C_1 the set of maximal complete subgraphs induced on the subsets of $A(v_1)$ the Cartesian product $\{v_1, v_2, \ldots, v_k\} \times C_1$ is precisely the set of all cliques of G containing one of the vertices $v_1, v_2, \ldots, v_k$. It is evident that this procedure could be extended so that the cliques of C_1 were also expressed as a set of Cartesian products each one being determined by a set of CS equivalent vertices and their common set of adjacent vertices defined on the subgraph induced on $A(v_1)$. As an example we may consider again the graph K(3,3,3) given in Fig. 2.1. The vertices 1,2, and 3 are CS equivalent and are each adjacent to vertices 4,5,6,7,8,9. In the subgraph induced on this latter set of vertices, the vertices 4,5,6 are CS equivalent and each is adjacent to vertices 7,8,9. Since the vertices 7,8,9 are isolates they are also CS equivalent. Thus all the cliques of the graph are given by the expression: $\{1,2,3\} \times \{\{4,5,6\} \times \{7,8,9\}\}$.

This is obviously a much more compact way of defining the twenty-seven cliques of K(3,3,3).

The primary drawback of implementing the technique just described is the paucity of vertices which are CS equivalent in an arbitrary graph. Instead, we shall implement a procedure which uses a weakened form of the definition of complete subgraph equivalence to group the vertices in a similar manner.

DEFINITION 3.2: Two vertices u and w are weakly CS equivalent if there exists a complete subgraph defined on some subset of vertices of G say $u, v_1, v_2, \ldots, v_j$ such that the subgraph induced on $w, v_1, v_2, \ldots, v_j$ is also complete.
It is clear from the definition that weakly CS equivalent vertices are not necessarily similar, and that complete subgraph equivalent vertices are weakly CS equivalent.

We now consider the properties of a set of weakly CS equivalent vertices defined in the following way. Let v_1 be an arbitrary vertex from V(G) and let $\{v_2, \ldots, v_j\}$ be a set of vertices also from V(G) such that each vertex v_i is adjacent to all vertices in $A(v_1)$. All the complete subgraphs containing v_1 including the cliques are induced on $\{v_1\} \cup A(v_1)$. If $\{w_1, w_2, \ldots, w_k\} \subseteq A(v_1)$ induces a complete subgraph then $v_1, w_1, w_2, \ldots, w_k$ also induces a complete subgraph. Further, since v_i is adjacent to every vertex in $A(v_1)$, it is adjacent to $w_1, w_2, \ldots, w_k$ and hence $\{v_i, w_1, w_2, \ldots, w_k\}$ induces a complete subgraph. Therefore v_1, v_i are weakly CS equivalent. Most

importantly it is the case that every clique of G with vertices from the set $\{v_i\} \cup A(v_1)$ is induced on $\{v_1, v_2, \ldots, v_j\} \cup A(v_1)$. In other words, given any vertex set $\{v_i, w_1, \ldots, w_k\}$ inducing a complete subgraph, all possible vertices which could be used to extend that vertex set so as to induce a larger complete subgraph must be contained in $\{v_1, v_2, \ldots, v_j\} \cup A(v_1)$.

If we denote by V_1 the vertex sets of all complete subgraphs which are maximal on the subgraph induced on $\{v_1, v_2, \ldots, v_j\}$, and denote by V_2 the vertex sets of all complete subgraphs which are maximal on the subgraph induced on $A(v_1)$, then the vertex sets determined by the Cartesian product $V_1 \times V_2$ induce complete subgraphs which are maximal on G.

This result has an alternative interpretation as a product of graphs. The join (see for example Harary [43]) of two graphs G_1 and G_2, denoted $G_1 + G_2$, is the graph G defined on $V(G_1) \cup V(G_2)$ such that every edge of G_1 or G_2 is an edge of G and for every vertex v in $V(G_1)$ and vertex w in $V(G_2)$, (v,w) is also an edge of G. Let C_1 be a complete subgraph which is maximal on the subgraph induced on $\{v_1, v_2, \ldots, v_j\}$, and let C_2 be a subgraph which is maximal on the subgraph induced on $A(v_1)$. Then $C_1 + C_2$ is a clique of G.

We illustrate the determination of the cliques of the graph by finding weakly CS equivalent vertices in the following example. Consider the graph of Fig. 2.9. It is evident by inspection that no pair of vertices exists which is

complete subgraph equivalent. Instead we define two sets of vertices by first choosing arbitrarily vertex 1 and defining one set to be A(1) = {2,6}. Now vertices 3 and 5 are also adjacent to {2,6} so the second set is {1,3,5}. By repeating this procedure on the subgraphs induced on the first of these vertex sets we discover that {2,6} can be separated into two sets expressible as a Cartesian product {2}X{6} corresponding to a complete subgraph of order 2. The vertex set {1,3,5} however consists of two components, an isolate 1, and a complete subgraph of order 2 defined on {3,5} and expressible as{3X5}. The complete subgraphs of the subgraph induced on {1,3,5} are induced on vertex sets {1} and {3}X{5} and hence some of the cliques of G are given by {{2}X{6}]X{1,{3 }X{5]} which corresponds to the cliques (261) and (2635). It is clear that not all the cliques have been found for we have not yet examined vertex 4. We therefore determine two new sets {3,5} and {2,4,6} in the same way, and the processing of their induced subgraphs yields {{3}X{5]}X {4,{2}X{6}} giving us the third subgraph (354).

This example illustrates the principal drawback of this procedure for enumeration, namely the generation of redundant cliques. We have encountered this type of problem in nearly all of the algorithms previously discussed. The most usual means of overcoming this problem has been to simply examine each vertex of the induced subgraph to see if there exists some vertex not in the set, yet adjacent to all vertices in the set. Such a mechanism is clearly not applicable in this case. Alternatively, Peay in his algorithm as originally

described, maintained all cliques in a stack which he could compare with newly determined maximal complete subgraphs to see whether it had been found before. Although more directly applicable to our situation, this method is not useful since we have no explicit representation for each clique with which to compare. In addition we must make use of a possibly exponential amount of storage. This difficulty is partly overcome in the following algorithm by making use of information pertaining to the current derivation path in the tree of derivations in a manner described in the next section.

SUBGRAPH EQUIVALENCE ALGORITHM

Two stacks are used in the algorithm. Stack 1 consists of all vertex sets derived in the development of the current derivation path except those from which the current set is derivable. Stack 2 consists of all vertex sets directly derivable from the last vertex set in the derivation path.

BLOCK A: Initialization procedure.

STEP1: Let V(G) be the set of all vertices in the graph and initially set both stacks to be empty.

STEP2: Call recursive procedure ENUM(V(G)) defined in BLOCK B. The order and adjacency matrix of G are defined globally to ENUM On return, stop.

BLOCK B: Recursive procedure ENUM(V(G)).

STEP1: Choose a vertex v in V(G) and define a vertex set F

equal to the union of v together with all vertices not adjacent to v.

STEP2: Choose a vertex w from F and define vertex sets H1 = V(G) ∩ A(w) and H2 equal to the set of all vertices in F adjacent to every vertex in H1.

STEP3: If the vertex set H1 ∪ H2 is contained in a vertex set previously defined during this iteration then go to STEP7.

STEP4: If the vertex set H1 ∪ H2 contains a previously defined vertex set during this iteration then replace that vertex set by H1 ∪ H2 on stack 2.

STEP5: Compare H1 ∪ H2 with all vertex sets generated during previous iterations in the development of the current derivation path other than those vertex sets from which H1 ∪ H2 was derived. If H1 ∪ H2 is contained in some previous such set then delete it from stack 2. Otherwise place the new set on stack 1.

STEP6: A new pair of vertex sets has been found. Stack 2 contains their union as well as the vertex w used to define them.

STEP7: Delete w from F. If F is not empty then go to STEP2.

STEP8: If no new vertex sets have been added to stack 2 this iteration then return.

STEP9 Choose a pair of sets H1 ∪ H2 from stack 2 together with their defining vertex w. Remove this set from stack 1.

STEP10: If H1 is empty then print vertex w and go to STEP14.

STEP11: Call recursive procedure ENUM(H2).

STEP12: Print "X". (The maximal complete subgraphs of H1 ∪ H2 will be given by the Cartesian Product of the results

upon return from calls in STEP11 and STEP13.

STEP13: Call recursive procedure ENUM(H1).

STEP14: Return H1 ∪ H2 and w to stack 1 but delete it from stack 2. If stack 2 not empty then go to STEP9.

STEP15: Return.

As previously described, the algorithm finds the cliques of the graph by determining sets of weakly CS equivalent vertices in a particular way. A new vertex set whose vertices have been partitioned into two sets of weakly CS equivalent vertices is determined by choosing a vertex v from a set F and defining the two blocks H1 and H2 of the partition according to STEP2. The set F consists of a vertex v and all vertices of the induced subgraph on the current set of vertices, V(G), under consideration not adjacent to v. This set insures that all cliques will be found and was employed in the Harary-Ross algorithm and the Bron-Kerbosch algorithm as well as our own sequential algorithm previously discussed in Chapter 2.

We are thus guaranteed of finding all the cliques and it is therefore only necessary to minimize the possibility of finding redundant cliques. As we have mentioned, this is not a simple problem because of the nature of the representation being exploited in our algorithm. The technique employed is to keep track of all vertex subsets from which a newly determined vertex subset could possibly be derived. To do this it is sufficient to keep track of only the initial nodes of all possible branches in the derivation tree which deviate from the path of derivations we have taken to reach the current

vertex subset under consideration. By definition all other vertex subsets derived during the execution will be contained in one of the vertex subsets represented by these nodes. In a derivation path of length k, let v_i be the defining vertex of the vertex set H1 $\cup$ H2 represented by a node on the derivation path at distance i from the root. It is evident that the maximum number of vertex subsets generated during the generation of set H1 $\cup$ H2 is $n_{i-1} - d(v_{i-1})$ where n_{i-1} is the number of vertices in the i-1st vertex subset in the path of derivations and v_{i-1} is its defining vertex. The maximum number of vertex subsets placed on stack 1 is thus $\sum_{i=1}^{k} (1+(n_{i-1} - d(v_{i-1})))$.

In Fig. 3.5 we illustrate the vertex subsets involved in such a sequence of derivations. Since our algorithm employs a depth before breadth technique of development, it is clear that stack 1 is not exponentially growing. Hence any gains in efficiency from such a representation will not be offset by inordinate storage requirements.

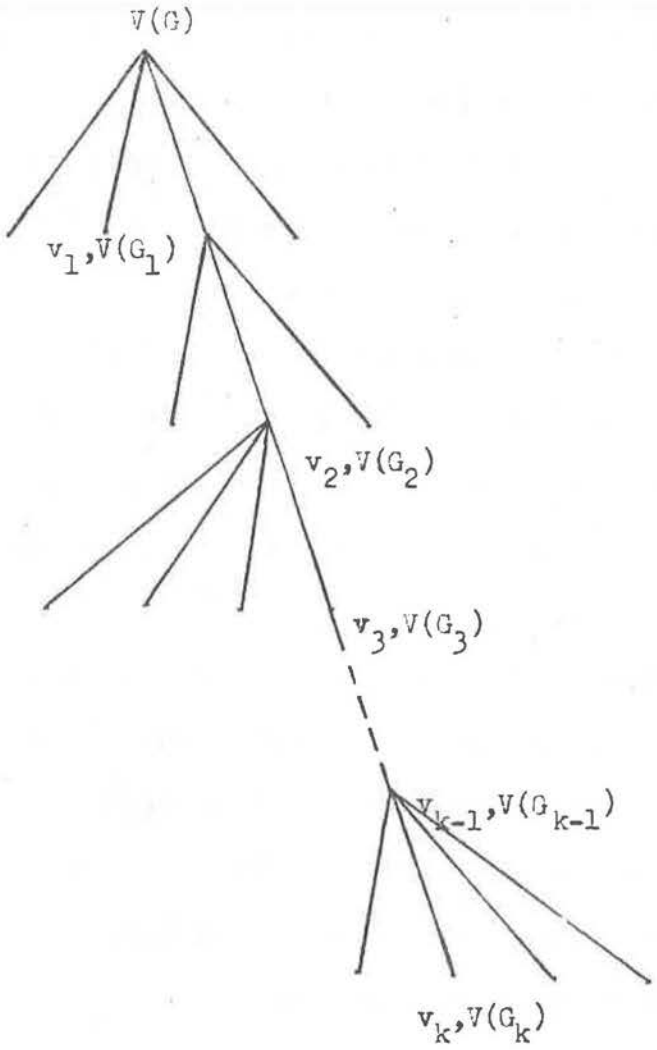


Fig. 3.5

A PATH IN THE DERIVATION OF NON-SIMILAR CLIQUES

It is difficult to assess the overall efficiency of this algorithm because the choice of a "best" defining vertex at each iteration is a non-deterministic procedure. Clearly, the worst case occurs when the choice that is made results in an explicit enumeration of every clique in the graph. In this situation, since the mechanism by which new vertex subsets are generated is similar to that of the Bron-Kerbosch or our sequential algorithm for the enumeration of cliques, a one-one correspondence can be made between the nodes of the derivation tree of this new algorithm with either of the previously discussed sequential algorithms.

Because a search of as many as $\sum_{i=1}^{k}(n_{i-1}-d(v_{i-1}))$ elements in a stack must be made (see Fig. 3.5) for each new vertex subset in addition to its generation, it is evident that the time required for one iteration will be longer than that required by the sequential method. Since we have employed the same techniques of our previous algorithm to the generation of new vertex subsets, the time required for one iteration is proportional to $\sum_{i=1}^{k}(n-d(v_i))\cdot T(n)$ where $T(n)$ is the time required for one iteration of the sequential algorithm of Chapter 2.

When, however, we can take advantage of the weak CS-equivalence of vertices to minimize the number of vertex subsets generated, the maximum efficiency is realized by the greatly reduced derivation tree. This is clearly evident for any complete k-partite graph $K(m_1,m_2,\ldots,m_k)$. Here, each block of m_i vertices corresponds to a set of complete subgraph

equivalent vertices, and hence also a set of weakly CS equivalent vertices. The derivation tree for $K(m_1,m_2,\ldots,m_k)$ using our new algorithm is linear, as only one vertex at each iteration defines a new subset. We illustrate such a derivation for $K(3^4)$ in Fig. 3.6, where the vertices of block V_i are labelled $(i-1)m+1$, $(i-1)m+2,\ldots,(i-1)m+m$, $i=1,2,3,4$, with $m = 3$.

For the complete k-partite graph $K(m^k)$, the number of nodes in the derivation tree determined by our algorithm is $mk+2(k-1)+1$ for $k > 1$ and $2(k-1)+1$ for $k = 1$.

Obviously then the derivation trees are smallest for complete k-partite graphs. As mentioned previously the worst case to be encountered occurs when there are no weakly CS equivalent vertices in the graph and consequently cliques are enumerated explicitly. The derivation tree may be used to determine the number of cliques in the graph. If we again examine Fig. 3.6, edges of the tree incident to a common vertex have been related by by the symbols "X" or "," according to whether the sets determined in that derivation can be combined in a Cartesian Product to obtain a subgraph of the original graph G. If not then their union (denoted by ",") is a subgraph of G. We illustrate this notation with the example of Fig. 3.7, a derivation of the cliques of Fig. 2.9 . Using such a notation, the cliques are given by the expression: {1 X {2 X 6}},{3 X {{2 X {5 X 6}},{4 X 5}}}.

From this example one can see that the clique (126) has been explicitly defined while those containing the vertex 3

(ie. (2356) and (345)) are grouped together. It is evident from Fig. 3.7 that there cannot be any edges not incident to the root of the derivation tree which are related by a "," for a graph whose cliques are all determined explicitly. Since the number of vertex subsets generated from the root is at most n-d(v) where v is a vertex of minimum degree, such a graph has fewer than n cliques. Hence all graphs having more than n cliques have some vertices which are weakly CS equivalent in the induced subgraph defined on some vertex subset; therefore some improvement over a sequential algorithm can often be obtained by reducing the number of vertex subsets that must be considered.

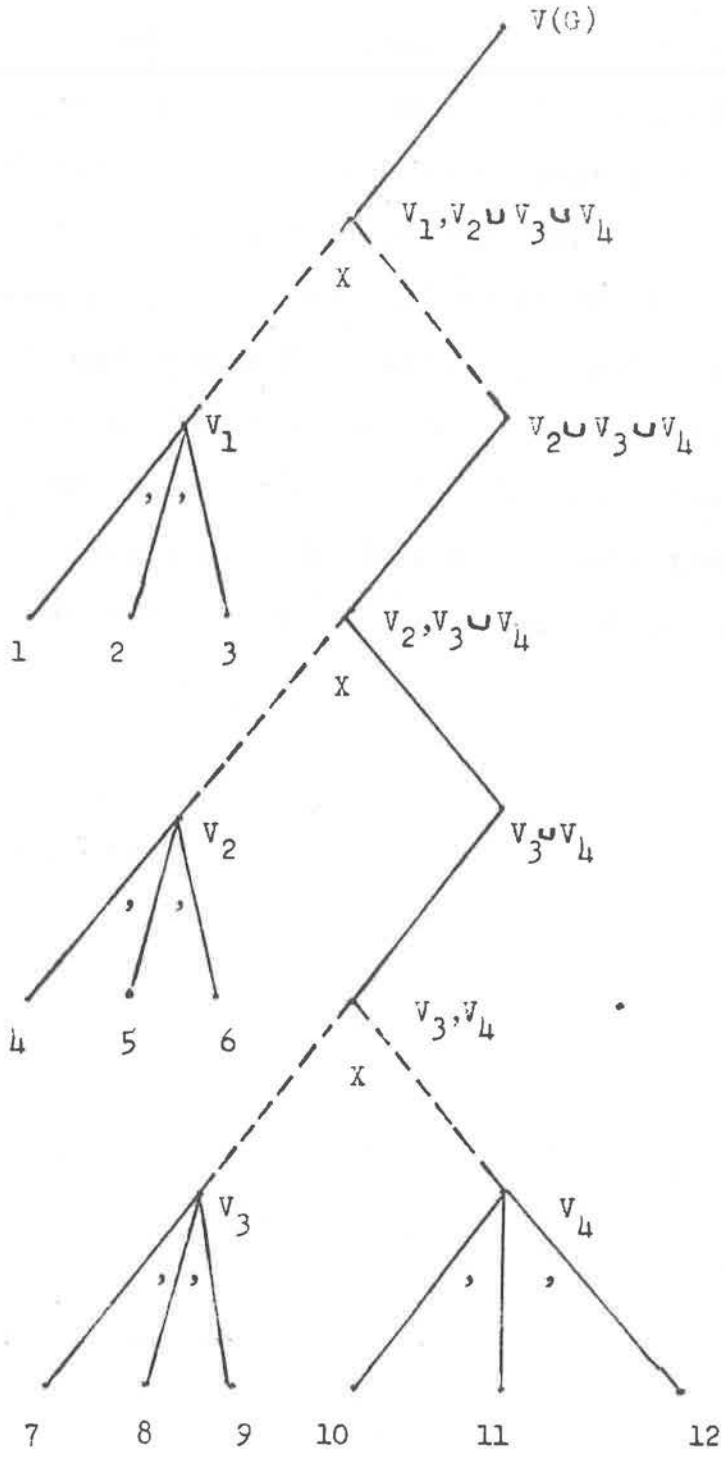


Fig. 3.6

DERIVATION TREE FOR K(3,3,3,3)

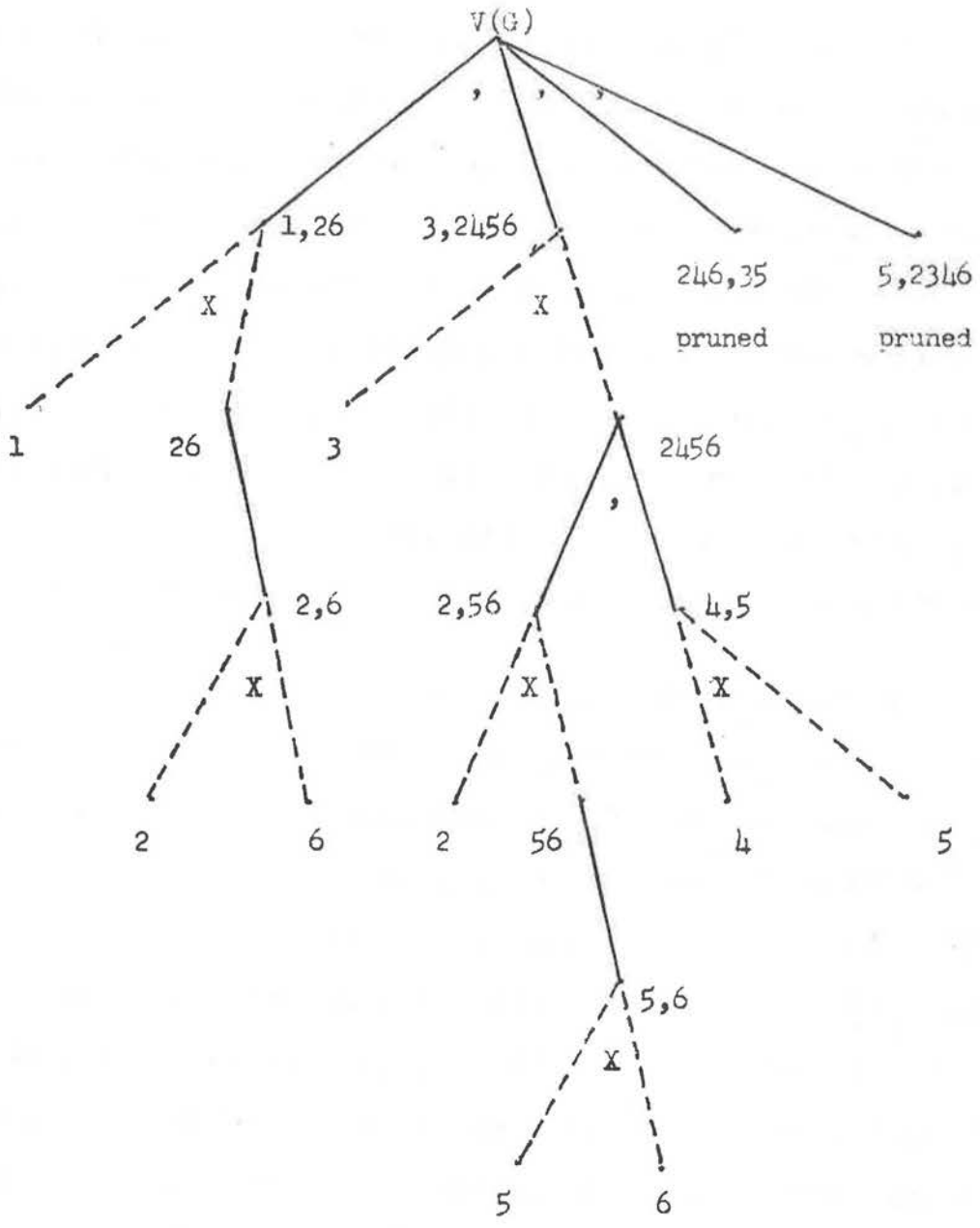


Fig. 3.7

DERIVATION TREE FOR FIG. 3.1

CHAPTER 4: EFFICIENT DETECTION OF CLIQUES

4.1 INTRODUCTION

In the previous two chapters we have explored some of the ways in which cliques can be detected in graphs. We have also examined how various properties associated with graphs might be used to improve the efficiency of such algorithms. The major observation to be made is that it is not at all clear how one might devise an efficient clique detection algorithm even to detect cliques of a particular order. In this chapter, however, a procedure for the detection of such cliques is proposed which can be proved to be an efficient algorithm for a particular class of graphs and for which no counterexample has yet been found for general k-partite graphs.

An important application of clique detection in graphs is motivated by the fact that it is possible to represent a well formed formula of the propositional calculus in disjunctive normal form as a k-partite graph where k is the number of conjuncts in the sentence. In the survey of Chapter 1 we mentioned briefly the efforts of Cook, Karp and Lawler, among others in developing a taxonomy of combinatorial problems. In particular we noted an important result of Cook's which relates the tautology problem to a number of other important combinatorial problems. An extensive list of these problems has been prepared by Karp [49]. We shall use his notation to define the concepts required in describing the equivalence of a k-partite graph to a well-formed formula in disjunctive

normal form.

4.2 CLIQUE DETECTION AND SATISFIABILITY

We turn now to a consideration of the "Satisfiability Problem" as defined by Karp [49] and its solution through the detection of cliques in a graph as suggested by Mowshowitz [63].

DEFINITION (KARP): The satisfiability problem is defined as follows: Given as input the clauses $C_1, C_2, \ldots, C_p$, of a well-formed formula in conjunctive normal form, does there exist a set $S \{x_1, x_2, \ldots, x_n; \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$ such that

a.) S does not contain a complementary pair of literals and

b.) $S \cap C_k \neq \Phi$ for $k=1,2,\ldots,p$.

We are thus given the well-formed formula $A \; C_1 \cap C_2 \cap \ldots \cap C_p$ and asked to decide whether or not it is satisfiable. To do this we convert its negation to disjunctive normal form. Suppose $\sim A$ is a tautology. Then for all possible assignments of truth values to the variables of $\sim A$, $\sim A$ is true and consequently A is false for all possible assignments. Therefore A is satisfiable if and only if $\sim A$ is not a tautology. It is the disjunctive normal form of $\sim A$ that we shall represent by a graph.

Let $S = D_1 \cup D_2 \cup \ldots \cup D_k$ be a sentence of the propositional calculus in disjunctive normal form with each conjunct $D_i = a_{i_1} \cap a_{i_2} \cap \ldots \cap a_{i_m}$ where a_{i_j} is a literal.

Define a k-partite graph G as follows. Each vertex in V(G) corresponds to a literal of S, there being as many vertices as there are literals of S. The unordered vertex pair (v_a, v_b) corresponding to literals a and b is an edge of G if and only if a is not the complement of b, and a and b are not both members of the same conjunct. Thus to each conjunct of S there corresponds a vertex block of G consisting of mutually non-adjacent vertices.

This representation can be used to determine whether or not S is a tautology. The decision rule is:

THEOREM 4.1: S is a tautology if and only if there does not exist a clique of order k in the corresponding graph G, k being the number of conjuncts in the disjunctive normal form.
Proof: A clique of order k in G exists if and only if there exists a selection of literals, one from each conjunct of S such that no literal and its complement are both contained in the selection. If such a selection exists then we can assign the value 0 (false) to each of the literals in the selection and hence negate the well formed formula. On the other hand if such a selection is not possible this corresponds to the fact that no such assignment to the literals of the well-formed formula can be made and hence it must be a tautology. QED.

The object of this chapter is to describe an algorithm which provides an efficient heuristic for determining whether a clique of order k exists in an arbitrary k-partite graph. Such an algorithm can then be employed through theorem 4.1 as

an efficient solution to the tautology problem.

Before presenting such an algorithm it is necessary to define and discuss a collection of vertex sets determined by the algorithm for a k-partite graph G which provides the mechanism for detecting the existence of cliques of order k. The importance of these vertex sets will be established in a subsequent theorem. First, however, we shall assume that we are given a k-partite graph G with its vertex set V(G) partitioned into k blocks $V_1, V_2, \ldots, V_k$ of mutually non-adjacent vertices.

We define $W_i^s(u,w)$, $i \leq s \leq k-2$ to be be a subset of block V_i associated with an edge (u,w) such that $W_m^t(u,w) \neq \Phi$, $m=1,2,\ldots,t$; $t=1,2,\ldots,s-1$.

If $i=s$ then $W_s^s(u,w) = V_s \cap A(u) \cap A(w)$ where $A(u), A(w)$ denote the adjacency sets of u and w respectively.

Else for $i < s$ $W_i^s(u,w)$ is the set of all vertices v_i in V_i such that

(a) $v_i \in W_i^{s-1}(u,w) \cap \left[\overbrace{}^{s}_{r=i+1} \left[\underbrace{}_{v_r \in W_r^s(u,w)} W_i^{r-1}(u,v) \cap W_i^{r-1}(w,v) \right] \right]$

(b) $W_j^{i-1}(u,v_i) \cap W_j^{i-1}(w,v_i) \neq \Phi$ for $j=1,2,\ldots,i-1$.

It is evident from the definition that for any particular value of s a family of sets associated with an edge (u,w) is determined in the order $W_s^s(u,w), W_{s-1}^s(u,w), \ldots, W_1^s(u,w)$. This order is a consequence of the fact that $W_i^s(u,w)$ is dependent upon $W_{i+1}^s(u,w), W_{i+2}^s(u,w), \ldots, W_s^s(u,w)$. A number of properties associated with this family of sets may be readily determined from the definition:

PROPERTY 1: $W_i^s(u,w) \subseteq W_i^{s-1}(u,w)$.

PROPERTY 2: $W_i^s(u,w)$ not empty implies that $W_{i+j}^s(u,w)$, j=1,i,...,s-i, not empty.

PROPERTY 3: Every vertex in $W_i^s(u,w)$ is adjacent to u and w.

4.4 CLIQUE DETECTION ALGORITHM

The algorithm proceeds by constructing the vertex sets $W_i^s(u,w)$, s=1,2,...,k-2 for each edge in the graph G. An edge (u,w) is deleted from the graph whenever $W_i^s(u,w)$ becomes empty for some i=1,2,...,s. After the sets $W_i^{k-2}(u,w)$, i=1,2,...,k-2, have been constructed for edges remaining in G connecting vertices in blocks V_{k-1} and V_k, and if at least one such edge remains, then the sets $W_i^s(u,w)$ are redefined by iterating the above procedure. These iterations continue until one of two conditions occurs:

(a) All edges have been eliminated from G.

(b) The latest iteration resulted in no further deletions.

The following theorem establishes that condition (a) implies G contains no complete subgraph of order k.

THEOREM 4.2: If G has a complete subgraph of order k then $W_1^{k-2}(u,w)$ is not empty for some edge (u,w) of G.

Proof: From the definition, the theorem is true for k = 3,4. Assume that for k = s it is the case that $K(1^s)$ is a subgraph

of G implies $W_1^{s-2}(u,w)$ is not empty for some edge (u,w) in $E(G)$, where u,w are in $V(K(1^s))$. Suppose further that $K(1^{s+1})$ is a subgraph of G where $V(K(1^{s+1})) = \{v_1,v_2,\ldots,v_{s-1},u,w\}$, v_i being a vertex from block V_i. Complete subgraphs of order s are defined on vertex sets:

$$V(K(1^s)) = \{v_1,v_2,\ldots,v_{s-2},u,w\}$$
$$V(K(1^s)) = \{v_1,v_2,\ldots,v_{s-2},v_{s-1},u\}$$
$$V(K(1^s)) = \{v_1,v_2,\ldots,v_{s-2},v_{s-1},w\}.$$

Therefore from the induction hypothesis v_1 is contained in each of $W_1^{s-2}(u,w)$, $W_1^{s-2}(u,v_{s-1})$, and $W_1^{s-2}(w,v_{s-1})$. Since $v_1,v_2,\ldots,v_{s-2},u,w$ are mutually adjacent, v_1 contained in $W_1^{s-2}(u,w)$ implies v_1 is contained in $\bigcap_{r=2}^{s-2}[W_1^{r-1}(u,v_r) \cap W_1^{r-1}(w,v_r)]$ for $r = 2,3,\ldots,s-1$. Hence by definition v_1 is contained in $W_1^{s-1}(u,w)$. QED.

The following lemmas and theorem show that if there are at most two vertices per vertex block of G with degrees greater than 0 after all possible edge deletions have been made then G contains a complete subgraph of order k.

LEMMA 4.1: Let $V_1,V_2,\ldots,V_k$ dencte the vertex blocks of a k-partite graph G and suppcse $|V_i| = 1$ for $i=1,2,\ldots,k$. Then $W_1^{k-2}(u,w)$ not empty implies $K(1^k)$ is contained in G.

Proof: By Property 2 $W_i^{k-2}(u,w)$ is not empty for $i=1,2,\ldots,k-2$. Since $|V_i| = 1$ $W_i^{k-2}(u,w) = V_i$. Let v_i be the vertex in block V_i. From the definition:

$$W_i^{k-2}(u,w) = \{v_i\} \cap \left[\bigcap_{r=i+1}^{k-2} [W_i^{r-1}(u,v_r) \; W_i^{r-1}(w,v_r)]\right]$$

implies v_i is adjacent to v_r for $r=i+1,i+2,\ldots,k-2$, and $i=1,2,\ldots,k-2$. Hence $\{v_1,v_2,\ldots,v_{k-2},u,w\}$ is a set of mutually adjacent vertices.

LEMMA 4.2: If G is a k-partite graph such that each vertex block has exactly two vertices, then for any edge (u,w) $W_1^{k-2}(u,w)$ not empty implies $K(1^k)$ is contained in G.

Proof: Let $k=5$ and suppose v_1 is a vertex in $W_1^3(u,w)$. Then there exists v_3 in $W_3^3(u,w)$ such that v_1 is contained in $W_1^2(u,w) \cap W_1^2(u,v_3) \cap W_1^2(w,v_3)$. It also follows from the definitions of $W_1^2(u,w)$, $W_1^2(u,v_3)$, and $W_1^2(w,v_3)$ that one can choose vertices x,y,z in V_2 such that complete subgraphs are induced on vertex sets $\{v_1,x,u,w\}$, $\{v_1,y,v_3,u\}$, and $\{v_1,z,v_3,w\}$.

Now since $|V_i| = 2$, either $x = y$, $x = z$, or $y = z$. If $x = y$ or $x = z$ then x is adjacent to v_3 and hence a complete subgraph is induced on $\{v_1,x,v_3,u,w\}$. If $y = z$ then y is adjacent to w and a complete subgraph is induced on $\{v_1,y,v_3,u,w\}$.

Assume the lemma is true for $k=s-1$ and suppose v_1 is a vertex in $W_1^s(u,w)$. Then by definition there exists v_s in $W_s^s(u,w)$ such that v_1 is contained in $W_1^{s-1}(u,w) \cap W_1^{s-1}(u,v_s) \cap W_1^{s-1}(w,v_s)$. Since v_1 is a member of each of the sets $W_1^{s-1}(u,w)$, $W_1^{s-1}(u,v_s)$, and $W_1^{s-1}(w,v_s)$, by the induction hypothesis one can find vertex sets X, Y, and Z such that complete subgraphs of order s+1 are defined on

$\{v_1,u,w\} \cup X$, $\{v_1,v_s,u\} \cup Y$, and $\{v_1,v_s,w\} \cup Z$. Each of the sets contains one vertex from each of the vertex blocks $V_2,V_3,\ldots,V_{s-1}$.

Let $S_1 = X \cap Y$, $S_2 = X \cap Z$, and $S_3 = Y \cap Z$. Then $S_1 \cup S_2 \cup S_3$ contains a vertex from each of blocks $V_2,V_3,\ldots,V_{s-1}$ since $|V_i| = 2$. Since each vertex in S_1 is adjacent to every vertex in X and Y, it is adjacent to every vertex in S_2 and S_3 and hence $S_1 \cup S_2$ and $S_1 \cup S_3$ are a set of mutually adjacent vertices. Similarly each vertex in S_2 is adjacent to every vertex in Z and hence $S_2 \cup S_3$ is also a set of mutually adjacent vertices. Hence $S_1 \cup S_2 \cup S_3$ induces a complete subgraph of order s-2. Further, by construction v_1,v_s,u, and w are adjacent to every vertex in $S_1 \cup S_2 \cup S_3$ and therefore G contains a complete subgraph of order s+2. QED.

THEOREM 4.3: If G is a k-partite graph with at most two vertices in each block, and if $W_1^{k-2}(u,w)$ is not empty for some edge (u,w), then G contains a complete subgraph of order k.

Proof: The result follows almost immediately from lemmas 4.1 and 4.2. If for some i, $W_i^{k-2}(u,w)$ contains only one vertex v_i then as a consequence of the definition v_i is adjacent to every vertex in every other set $W_j^{k-2}(u,w)$, $i \neq j$. Suppose there are $r \leq k-2$ sets having 2 vertices. By the method in lemma 4.2 we can find a complete subgraph of order r+2 defined on vertices chosen from these sets. By the remark above, each vertex in a set which it is the sole member is adjacent to all other vertices and therefore G contains a complete subgraph of

order k. QED.

Finally, we note that the following clique detection algorithm could be modified to work for all k-partite graphs. If condition (b) occurs and if the condition of theorem 4.3 is not satisfied, then a clique enumeration procedure can be applied to the subgraph of G which remains after no further edge deletions occur. This method has been implemented for verification purposes and the results are summarized in Chapter 5.

CLIQUE DETECTION ALGORITHM

Let G be a k-partite graph with blocks $V_1, V_2, \ldots, V_k$. Denote by A(u) the adjacency set of vertex u.

STEP1: Set s to 1. Define graph H_0 equal to G.

STEP2: If $E(H_{s-1})$ empty then stop--$K(1^k)$ is not in G.

STEP3: Choose an edge (u,w) in $E(H_{s-1})$ where u and w are vertices in $V(H_{s-1}) - V_s$.

STEP4: Set $W_s^s(u,w)$ to $V_s \cap A(u) \cap A(w)$.

STEP5: If s = 1 then go to STEP16.

STEP6: set i to s-1.

STEP7: Set r to i+1. Set P to be empty.

STEP8: Choose a vertex v in $W_r^s(u,w)$.

STEP9: If (u,v) and (w,v) are edges of H_{s-1} and $W_i^{r-1}(u,v) \cap W_i^{r-1}(w,v)$ not empty then go to STEP12.

STEP10: Set $W_r^s(u,w)$ to $W_r^s(u,w) - \{v\}$.

STEP11: If $W_r^s(u,w)$ empty then set $W_1^s(u,w)$ to be empty and go

to STEP16.

STEP12: Set P to P $[W_i^{r-1}(u,v) \cap W_i^{r-1}(w,v)]$.

STEP13: If some v in $W_r^s(u,w)$ has not been examined then go to STEP8.

STEP14: If P not empty then set $W_i^s(u,w)$ to $W_i^{s-1}(u,w) \cap P$. Otherwise set $W_1^s(u,w)$ to be empty and go to STEP16.

STEP15: Set i to i-1. If $i \geq 1$ then go to STEP7.

STEP16: If $W_1^s(u,w)$ has not been computed for all edges (u,w) in $E(H_{s-1})$, where u,w are in $V(H_{s-1})-V_s$, then go to STEP3.

STEP17: Define graph $H_s \subseteq H_{s-1}$ as follows:

a.) $V(H_s)$ is set to $V(H_{s-1}) - V_s$,

b.) (u,w) is an edge of $E(H_s)$ if and only if (u,w) is an edge of $E(H_{s-1})$ and $W_1^s(u,w)$ is not empty.

STEP18: Set s to s+1. If $s \leq k-2$ then go to STEP2.

STEP19: Let G' be the subgraph of G such that v is in V(G') if and only if there exists (u,w) in $E(H_{s-1})$ such that v is in $W_i^{k-2}(u,w)$ for some $i = 1,2,\ldots,k-2$. If $G' \neq G$ then let $G = G'$ and go to STEP1.

STEP20: If G' has at most 2 vertices in each block of degree > 0 then $K(1^k)$ is contained in G; else a clique must be verified by enumeration.

As an example, consider the graph of Fig 4.1. We summarize the results of the algorithm in TABLE 4.1. The subgraph defined by STEP19 after 1 iteration is the complete subgraph of order 5. The second iteration defines sets for this graph as indicated in the table. Vertices which are deleted according to STEP10 are indicated in parentheses. By STEP19, the algorithm terminates after the second iteration.

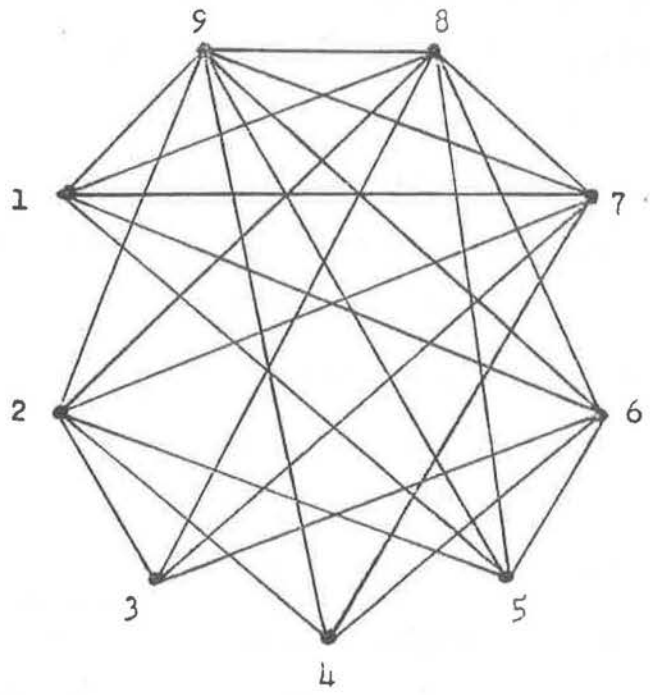


Fig. 4.1

GRAPH CONTAINING SUBGRAPH $K(1^5)$

ITERATION	EDGE	w_1^1	w_2^2	w_1^2	w_3^3	w_2^3	w_1^3
	(36)	Φ					
	(37)	2					
	(38)	2					
	(46)	Φ					
	(47)	2					
	(49)	2					
1	(56)	1					
	(58)	1,2					
	(59)	1,2					
	(68)	1	(3),5	1			
	(69)	1	(4),5	1			
	(78)	1,2	3	2			
	(79)	1,2	4	2			
	(89)	1,2	5	1,2	6,(7)	5	1
	(56)	1					
	(58)	1					
2	(59)	1					
	(68)	1	5	1			
	(69)	1	5	1			
	(89)	1	5	1	6	5	1

TABLE 4.1

RESULTS OF THE ALGORITHM FOR FIG.4.1

4.4 TIMING CONSIDERATIONS

Let G be a complete k-partite graph with m vertices in each block V_i i = 1,2,...,k. For such a graph any choice of vertices $v_1, v_2, \ldots, v_k$ with v_i a member of V_i is the vertex set of some $K(1^k)$ contained in G. Further every edge in G is a member of some $E(K(1^k))$. G therefore constitutes the "worst" k-partite graph the algorithm can encounter.

During iteration s all edges defined over $V_{s+1} \cup V_{s+2} \cup \ldots \cup V_k$ must be examined. There are $\frac{m^2}{2}(k-s)(k-s-1)$ such edges. For each edge (u,w) we must compute $W_s^s(u,w), W_{s-1}^s(u,w), \ldots, W_1^s(u,w)$. $W_i^s(u,w)$ can be determined in one intersection (that of rows u and w of the adjacency matrix treated as n digit binary numbers), while $W_i^s(u,w)$ is computed from the definition as follows. $\bigcup_{v_r \in W_r^s(u,w)} W_i^{r-1}(u,v_r) \cap W_i^{r-1}(w,v_r)$ is computed in m intersections, m-1 unions and m tests for emptyness for each r = i+1, i+2, ..., s. The total computation time for $W_i^s(u,w)$ is therefore 1+(s-i)(3m-1), so to compute all s sets requires $1 + \sum_{k=1}^{s-1} ((s-i)(3m-1) + 1)$ steps. The total number of steps at iteration s for all edges is thus given by:

$$\left(\frac{m^2}{2}(k-s)(k-s-1)\right)\left(1 + \sum_{i=1}^{s-1} ((s-i)(3m-1) + 1\right)$$

Since there are k-3 iterations requiring these computations (iteration 1 only computes $W_1^1(u,w)$ for $\frac{m^2}{2}(k-1)(k-2)$ edges), the total number of steps in the computation is:

$$\frac{m^2}{2}(k-1)(k-2) + \sum_{s=2}^{k-2} \frac{m^2}{2}(k-s)(k-s-1)\frac{s}{2}(s-1)(3m-1) + s$$

$$= \left[\frac{m^2(k^2-3k+2)}{4}\right]\cdot\left[\frac{3m-1}{30}k^3 + \frac{m+1}{2}k^2 - \frac{3m-38}{5}k + \frac{3(3m-1)}{5}\right]$$

Hence an iteration of the algorithm is $O(k^5)$ with leading coefficient $\frac{m^2(3m-1)}{120}$.

Since at least one edge must be deleted during each iteration of the procedure, a crude upper bound on the number of iterations is given by the number of edges of the graph. In general it is difficult to obtain a sharper bound although empirical tests on a large sample of graphs yielded none requiring more than two iterations to reach a decision. In any case, the time required to execute the procedure remains bounded by a polynomial in k.

4.5 STORAGE CONSIDERATIONS

Let G be the graph described in our discussion of timing considerations. Both the adjacency matrix of G and the storage required for saving the W-sets place the greatest demand on space. Each $W_i^s(u,w)$ and row of the adjacency matrix can be represented by bit strings of length m and mk respectively. Since edges with incident vertices in V_r will not be considered after iteration r-1, such edges require storage for r-1 terms $W_{r-1}^{r-1}, W_{r-2}^{r-1}, \ldots, W_1^{r-1}$. An examination of the complete k-partite graph G with block size m reveals that m^2 edges will have k-2 such terms computed for them (those with one vertex in V_{k-1} and one in V_k), $2m^2$ edges will have k-3 such terms, ..., $(k-2)m^2$ edges will have only one such term. Since only the most recent values of any such term are ever required the maximum number of terms is $m^2 \sum_{i=1}^{k-2} i(k-i-1) = \frac{m^2}{2}\binom{k}{3}$. The

total storage requirement in bits for these items is thus $(mk)^2 + \frac{m^3}{2}\binom{k}{3}$.

4.6 IMPLICATIONS OF THE PROCEDURE

We have established that the procedure is an efficient algorithm for detecting k-cliques in graphs having at most two vertices per block. When used as a tautology testing procedure, this agrees with Cook's result[11] that well-formed formulae having at most two literals per clause can be determined to be tautologies in polynomial time. Moreover, if graphs having more than two vertices per block, each of degree > 0, can be reduced to graphs having at most two per block, then the procedure is still guaranteed to detect the existence of k-cliques in polynomial time. Thus, the algorithm accepts a larger class of well-formed formulae (represented as graphs) than the Davis-Putnam procedure [66].

In light of the work of Cook and Karp, the procedure provides a mechanism for greatly reducing the number of well-formed formulae that might require an exponential solution to the tautology problem. The construction of a specific counterexample might help to resolve the questions raised by Cook and Karp concerning the exponential nature of the tautology problem. However, finding such a counterexample remains an open problem.

CHAPTER 5: EMPIRICAL OBSERVATIONS AND SUMMARY

5.1 INTRODUCTION

As has been observed in Chapter 2, the clique enumeration algorithms discussed there all operate in essentially the same way, namely the development of a derivation tree from a node representing the initial vertex set of the graph to nodes representing the cliques of the graph. The thesis of Chapter 2 is that the size of the derivation tree developed by each algorithm applied to a Moon-Moser graph together with the order of computation for one iteration can be used as a measure of its efficiency. This efficiency therefore depends on the technique employed to develop new nodes of the derivation tree in as much as this procedure determines their total number. The algorithms examined in Chapter 2 all use different methods to generate new vertex subsets with the exception of the Bron-Kerbosch algorithm which was seen to develop the same derivation tree as the Reduced Redundancy algorithm. The purpose of comparison in this Chapter is to compare the actual performance of different methods for developing the derivation tree with the results of Chapter 2.

The same set of data used to test the clique enumeration algorithms was also used to test the efficiency of the clique detection procedure of Chapter 4 in order to determine how much better its performance was than employing an ordinary enumeration algorithm.

5.2 THE TEST DATA

In order to determine the performance of the algorithms, random graphs of various edge densities were generated for graphs having 9, 12, 18, and 21 vertices. Each algorithm was then run on the test data. For each graph and each algorithm the following statistics were recorded:

a.) The time required to find all the cliques,

b.) The number of vertex sets examined,

c.) The number of cliques found.

Sixteen test graphs were generated and their orders, edge densities, and number of cliques are given in TABLE 5.1. The results of applying each algorithm to this set of graphs is given in TABLE 5.2 (time in seconds) while the actual implementations used may be found in APPENDIX E. Due to excessive computation time, the algorithms of Harary-Ross and Peay were not applied to some of the graphs. Because of the features of dynamic storage allocation and bit string manipulation, PL/1 was used as the programming language for the implementations of these algorithms. The programs were run on an IBM 360/67 Duplex system operating under MTS. The actual graphs may be found in APPENDIX C.

graph	vertices	edge density	cliques
1	9	0.4	8
2	9	0.6	10
3	9	0.8	17
4	9	1.0	27
5	12	0.4	12
6	12	0.6	19
7	12	0.8	31
8	12	1.0	81
9	18	0.4	34
10	18	0.6	39
11	18	0.8	69
12	18	1.0	729
13	21	0.4	43
14	21	0.6	58
15	21	0.8	144
16	21	0.9	392

TABLE 5.1

CHARACTERISTICS OF THE TEST GRAPHS

GRAPH	HARARY-ROSS ALGORITHM		PEAY'S ALGORITHM		BONNER'S ALGORITHM		R. R. ALGORITHM	
	TIME	NODES	TIME	NODES	TIME	NODES	TIME	NODES
1	0.51	15	0.20	36	0.15	27	0.20	20
2	0.71	19	0.48	46	0.17	34	0.23	24
3	1.28	33	0.47	49	0.22	48	0.28	31
4	1.97	53	0.71	79	0.27	64	0.42	40
5	1.85	23	0.64	62	0.25	55	0.35	37
6	2.62	37	1.11	103	0.36	77	0.50	51
7	4.75	61	0.97	99	0.70	142	0.68	65
8	11.80	161	2.45	241	0.97	256	1.18	121
9	8.06	67	2.48	182	0.55	117	0.85	78
10	9.48	79	2.91	221	0.71	143	1.36	91
11	18.67	137	11.13	655	2.77	465	2.01	150
12	---	---	---	---	15.31	4096	14.03	1093
13	13.91	85	3.32	231	0.73	149	1.54	108
14	16.32	115	6.12	444	1.62	354	2.35	167
15	---	---	21.75	1490	4.80	1177	5.36	374
16	---	---	---	---	15.03	4023	11.01	688

TABLE 5.2

COMPARISON OF ALGORITHMS

5.3 DISCUSSION OF THE RESULTS

The results of running each algorithm on the test data of TABLE 5.1 were used to verify the predictions made for the size of the derivation tree. For this purpose, the graphs numbered 4, 8, and 12 were examined as they correspond to Moon-Moser graphs (see page 13) on 9, 12, and 18 vertices respectively. Because of the slowness of the Harary-Ross algorithm and Peay's algorithm for the smaller graphs no attempt was made to obtain such results for the Moon-Moser graph on 18 vertices.

For the test cases, the Harary-Ross algorithm generated the fewest nodes of the derivation tree in finding the cliques of a graph yet performed more poorly than Bonner's algorithm which generated the largest derivation trees. This is a consequence of the fact that the method used by Harary and Ross to generate new vertex subsets while being very selective is also very time consuming as was seen in our analysis of this algorithm in Chapter 2. On the other hand, Bonner sacrifices efficient node generation in the derivation tree for a simple means of defining new vertex subsets. In spite of its defects as observed by Augustson and Minker [5], this method appears to be very successful particularly with small graphs as one would expect, since for such a graph the size of the derivation tree does not yet dominate the computation. This hypothesis is further supported by the observation that the Reduced Redundancy algorithm appears to be most competitive with Bonner's algorithm for graphs of high edge

density. Such graphs have large numbers of cliques and hence their derivation trees will be large.

Peay's algorithm performed significantly better than the Harary-Ross algorithm and would probably have been more competitive with the other algorithms if the size of derivation tree generated were reduced. Such a modification seems feasible if one were to employ a technique of examining all the non-adjacent vertices associated with a particular vertex at one time rather than step-by-step. This is an approach similar to that taken in the Reduced Redundancy algorithm.

If the order of computation for one iteration multiplied by the number of nodes in the derivation tree of $K(3^k)$ derived in Chapter 2 is used as a rough measure of relative efficiency, then good agreement is obtained with the empirical results. Although such an estimate does not indicate accurately how much better one algorithm is than another, the difference in magnitudes of these values, particularly with large graphs, provides some guide in choosing the most efficient algorithm.

The inefficiency of clique enumeration algorithms for finding the existence of a maximal complete subgraph of order k in a k-partite graph is revealed by the results given in TABLE 5.3 where we compare the computation time of the decision procedure of Chapter 4 applied to the graphs of TABLE 5.1 with the best time available for the Reduced Redundancy algorithm. While it is true that such enumeration procedures

could be modified to terminate when a k-clique had been discovered, this would not preclude the possibility of such cliques being discovered rather late in the enumeration.

GRAPH	ENUMERATION ALGORITHM TIME	DECISION ALGORITHM TIME
1	0.15	0.06
2	0.17	0.07
3	0.22	0.09
4	0.27	0.11
5	0.25	0.15
6	0.36	0.16
7	0.68	0.20
8	0.97	0.28
9	0.55	0.64
10	0.71	0.77
11	2.01	1.05
12	14.03	4.28
13	0.73	0.82
14	1.62	0.89
15	4.80	1.54
16	11.01	3.98

TABLE 5.3

COMPARISON OF DETECTION AND ENUMERATION

5.4 SUMMARY

The principal goal of this thesis has been to examine ways of detecting cliques in graphs. In particular, we have sought to derive an efficient algorithm for determining the existence of a clique of order k in a k-partite graph. This goal was achieved in Chaper 4 for a subset of all graphs by adopting a different approach to the problem than that offered in Chapters 2 and 3. In these chapters we saw that it was unlikely that we could solve our problem by employing any clique enumeration algorithm. This was because such algorithms could be compared to tree searching processes which are known to be inefficient procedures.

In Chapter 3 we attempted to explcit some properties of graphs which would allow us to group "similar" vertices together so that we would not have to examine each vertex in such a group individually. Two kinds of "similarity" were discussed: graph theoretic similarity, and complete subgraph equivalence. The latter offered some possibility of improvement because of the concise notational representation. However no solution was found for avoiding the problem of multiply defining cliques although an attempt was made to minimize such behavior. Further, it should be observed that the new type of notation while concise, does not readily display either the cliques or their orders.

As the number of clique enumeration algorithms in the literature increases, it is useful to carry out some empirical comparison of the performance of these procedures. It has been

our observation that the implementations of several of these algorithms have been extremely sensitive to modifications which improve their efficiency. Some of these modifications have been discussed previously and were included in the implementations. The efficiency of these implementations was examined in the previous section. While we have been able to suggest improvements to the algorithms discussed in this thesis, such changes have not really changed the basic approach and as a consequence their computation times remain exponential. Therefore empirical tests of such procedures are possible only for moderately large graphs. For very large graphs, determination of the size of the derivation tree provides a more useful and less expensive method for assessing the performance of enumeration algorithms.

In Chapter 4, an efficient algorithm was defined which detects the existence of k-cliques in certain k-partite graphs. Such graphs have the property that they can be reduced by the algorithm to graphs having at most two vertices of degree > 0 in each vertex block. The work of Cook and Karp suggests (but does not imply) that the algorithm will not work for all k-partite graphs. Proving this may help in characterizing why the tautology problem does not appear to be solvable in polynomial time.

BIBLIOGRAPHY

1 Abraham, C.T. "An application of clustering techniques to minimizing the number of interconnections in electrical assemblies " Some Problems In Information Science (M. Kochen ed.), 1965 pp. 252-265

2 Abraham, C.T. "Techniques for thesaurus organization and evaluation " Some Problems In Information Science (M. Kochen ed.), 1965, pp. 131-150

3 Abraham, C.T. "Graph theoretic techniques for the organization of linked data " Some Problems In Information Science (M. Kochen ed.), 1965, pp. 229-251

4 Andrasfai, B. "New proof of a graph theoretic theorem of Turan " Magyar Tud. Akad. Mat. Kutato. Int. Kozl 7(1962) pp. 193-196

5 Augustson, J. Gary, Minker, Jack "An analysis of some graph theoretical cluster techniques " J. Assoc Computing Machinery 17(1970) pp. 571-588

6 Bonner, R.E. "On some clustering techniques " IBM J. Research And Development 8(1964) pp. 22-32.

7 Boyle, R.P. "Algebraic systems for normal and hierarchichal sociograms " Sociometry 32(1969) pp. 99-119

8 Bron, Coen, Kerbosch, J.A.C.M. "Finding all cliques of an undirected graph " Communications Assoc. Computing Machinery to appear

9 Cartwright, D., Harary, F. "Structural balance: a generalization of Heider's theory " Psych. Rev. 63(1956) pp. 146-153

10 Coleman, J.S., MacRae, D. (jr.) "Electronic processing of sociometric data for groups up to 1000 in size " Amer. Sociological Rev. 25(1960) pp. 722-727

11 Cook, S.A. "The complexity of theorem-proving procedures " Third ACM Symposium On The Theory Of Computing 1970, pp. 151-158

12 Corneil, D.G. "Graph isomorphism " Department Of Computer Science, University Of Toronto Ph.D thesis, 1968

13 Corneil, D.G., Gotlieb, C.C. "An efficient algorithm for graph isomorphism " J. Assoc. Computing Machinery 17(1970) pp. 51-64

14 Culik, K. "On the chromatic decompositions and chromatic numbers of graphs " Spisy Prirod. Fak. Univ. Brno. 1959 pp. 177-185

15 Davis, J.A. "Clustering and structural balance in graphs " Human Relations 20(1967) pp. 181-187.

16 Dirac, G. "Extensions of Turan's theorem on graphs " Acta. Math. Acad. Sci. Hungar. 14(1963) pp. 417-422

17 Dirac, G. "On complete subgraphs and complete stars contained as subgraphs in graphs " Math. Scand. 12(1963) pp. 39-46

18 Dirac, G. "Extensions of the theorems of Turan and Zarankiewicz " Proc. Symp. Smolenice 1963 pp.127-132

19 Dirac, G. "Chromatic number and topological complete subgraphs " Canad. Math. Bull. 8(1965) pp.711-715

20 Doreian, P. "A note on the detection of cliques in valued graphs " Sociometry 32(1969) pp. 237-242.

21 Erdos, P. "Remarks on a theorem of Ramsey " Bull. Res. Council Israel Sect. F 7(1957-1958) pp. 21-24

22 Erdos, P. "Graph theory and probability I " Canad. J. Math. 11(1959) pp.34-38

23 Erdos, P. "Graph theory and probability II " Canad. J. Math. 13(1961) pp. 346-352

24 Erdos, P. "On a theorem of Rademacher-Turan " Ill. J. Math. 6(1962) pp 122-127

25 Erdos, P. "On the number of complete subgraphs contained in certain graphs " Magyar Tud. Akad. Mat. Kutato. Int. Kozl. 7(1962) pp. 459-464

26 Erdos, P. "On circuits and subgraphs of chromatic graphs " Mathematika 9(1962) pp. 170-175

27 Erdos, P. "On complete topological subgraphs of certain graphs " Ann. Univ. Sci. Budapest Eotvos Sect. Math. 7(1964) pp.143-149

28 Erdos, P. "On the number of triangles contained in certain graphs " Canad. Math. Bull. 7(1964) pp. 53-56

29 Erdos, P. "Some remarks on Ramsey's theorem " Canad. Math. Bull. 7(1964) pp. 619-622

30 Erdos, P. "On cliques in graphs " Israel J. Math. 4(1966) pp. 233-234.

31 Festinger, L. "Time analysis of sociograms using matrix algebra " Human Relations 2(1949) pp. 153-158.

32 Folkman, J. "Regular line symmetric graphs " J. Combinatorial Theory 3(1967) pp.215-232

33 Forsyth, E., Katz, L. "A matrix approach to the analysis of sociometric data: preliminary report " Sociometry 9(1946) pp. 340-347.

34 Gill, A. "Analysis of nets by numerical methods " J. Assoc. Computing Machinery. 7(1960) pp. 251-254

35 Giraud, G. "An upper bound for Ramsey number (5,5) " C.R. Acad. Sci. Paris 265(1968) pp. 809-811

36 Gotlieb, C.C., Kumar, S. "Semantic clustering of index terms " J. Assoc. Computing Machinery 15(1968) pp. 493-513

37 Graver, J., Yackel, J. "An upper bound for Ramsey numbers " Bull. Amer. Math. Soc. 72(1966) pp. 1076-1079

38 Graver, J., Yackel, J. "Some graph theoretic results associated with Ramsey's theorem " J. Comb. Theory 4(1968) pp. 125-175

39 Hajnal, A., Suranyi, J. "On the decomposition of graphs into complete subgraphs " Ann. Univ. Sci. Budapest. Eotvos Sect. Math. 1(1958) pp. 113-121

40 Harary, F., Norman, R.Z. Graph Theory As A Mathematical Model In Social Science Ann-Arbor: Institute for Social Research, 1953

41 Harary, F., Ross, I. "A procedure for clique detection using the group matrix " Sociometry. 20(1957) pp. 205-215.

42 Harary, F. "Graph theoretic methods in the management sciences " Management Science 5(1959) pp. 387-403

43 Harary, F. Graph Theory Addison-Wesley, Reading, Mass., 1969

44 Harary, F. "Graph theory as a structural model in the social sciences " Graph Theory And Its Applications Bernard Harris (ed.), 1970, pp. 1-16

45 Hubbell, C.H. "An input-output approach to clique identification " Sociometry. 28(1965) pp. 377-399.

46 Ilzinia, I. "Finding the cliques of a graph " Avtomat. I. Vychisl. Tekn. #2(1967) pp. 7-11.

47 Johnson, S.C. "Hierarchichal clustering schemes " Psychometrika 32(1967) pp. 241-254

48 Kalbfleisch, J. "On an unknown Ramsey number " Mich. Math. J. 13(1966) pp385-392

49 Karp, R.M. "Reducibility among combinatorial problems " Proceeding Of The IBM Conference On The Complexity Of Computations Plenum Press, New York, 1972

50 Kochen m. Some Problems In Information Science. Scarecrow Press, new york, 1965.

51 Lawler, E.L. "Electrical assemblies with minimum number of intersections " IRE Trans. EC-11(1962) pp. 86-88

52 Lawler, E.L. "Polynomial bounded and (apparently) non polynomial bounded matroid computations " Proceedings Of The NYC-CNR Symposium On Combinatorial Algorithms (to appear) Prentice-Hall, New York, 1972

53 Lawler, E.L. "Matroids with parity conditions: a new class of combinatorial optimization problems " Mathematical Programming submitted 1972

54 Luce, R.D., Perry, A.D. "A method of matrix analysis of group structure " Psychometrika. 14(1949) pp. 95-116.

55 Luce, R.D. "Connectivity and generalized cliques in sociometric group structure " Psychometrika. 15(1950) pp. 169-190.

56 Meetham, A.R. "Algorithm to assist in finding the complete subgraphs of a given graph " Nature 211(1966) p. 105

57 Meetham, A.R. "Graph separability and word grouping " Proc. Assoc. Comp. Mach. 21st National Conference 1966 pp. 513-514

58 Mendelson, E. Introduction To Mathematical Logic Van Nostrand, Princeton, N.J.,1964.

59 Moon, J.W. "On the number of complete subgraphs of a graph " Canad. Math. Bull. 8(1965) pp. 831-834

60 Moon, J.W., Moser L. "On cliques in graphs " Israel J. Math. 3(1965) pp. 23-28.

61 Moon, J. "On independent complete subgraphs in a graph " Canad. J. Math. 20(1968) pp. 95-102

62 Moreno, J.L. Who Shall Survive? A New Approach To The Problem Of Human Inter-Relations Beacon House, New York, 1934

63 Mowshowitz, A., (private communication)

64 Mulligan, G.D. "Algorithms for finding cliques of a graph " Department Of Computer Science, University Of Toronto Technical report No. 41, May 1972

65 Mulligan, G.D., Corneil, D.G. "Corrections to Bierstone's algorithm for generating cliques " J. Assoc. Computing Machinery 16(1972) pp. 244-247

66 Putnam, H., Davis, M. "A computing procedure for quantification theory" J. Assoc. Computing Machinery 7(1960) pp. 201-215

67 Nordhaus, E.A. "On the density and chromatic numbers of graphs " Lecture Notes In Mathematics. 110(1969) pp. 245-249.

68 Peay, E.R. (jr.) "An iterative clique detection procedure " Mich. Math. Psych. Prog. 4(1970).

69 Peay, E.R. (jr.) "Nonmetric grouping: clusters and cliques " Mich. Math. Psych. Prog. 5(1970)

70 Ramsey, F. "On a problem of formal logic " Proc. Lond. Math. Soc. 30(1930) pp. 264-286

71 Rose, M.J. "Classification of a set of elements " Computer J. 7(1964) pp. 208-211.

72 Sauer N. "A generalization of a theorem of Turan " J. Comb. Theory 10(1971) pp.109-112

73 Sims, Charles C. "Graphs and finite permutation groups " Math. Zeitschr. 95(1967) pp 76-86

74 Spencer, J.H. "On cliques in graphs " Israel J. Math. 9(1971) pp. 419-421.

75 Turan, P. "Eine extremalaufgabe aus der graphenthecrie " Mat. Fiz. Lapok 48(1941) pp. 436-452

76 Turan, P. "On the theory of graphs " Colloq. Math. 3(1954) pp. 19-30

77 Turner, James "Point-symmetric graphs with a prime number of points " J. Combinatorial Theory 3(1967) pp. 136-145

78 Weiss, R.S., Jacobsen E. "A method for the analysis of the structure of complex organizations " Amer. Sociological Review. 20(1955) pp. 661-668.

79 Zelinka, B. "On the number of independent complete subgraphs " Publ. Math. Debrecen. 13(1966) pp. 95-97

APPENDIX A

operation	symbol	time constant
STORE	<--	t_1
PUSH,POP	push,pop	t_2
ADD,SUBTRACT	+,-	t_3
COMPARE	:	t_4
MULTIPLY	•	t_5
UNION,INTERSECTION	$\cup$, $\cap$	t_6
COMPLEMENT	-	t_7
SUBSTRING	substr	t_8
INDEX	index	t_9
PRINT	print	t_{10}

LIST OF OPERATIONS

APPENDIX B

```
*****************************************************************
*                                                               *
*        THE MODIFIED HARARY-ROSS ALGORITHM                     *
*                                                               *
*                                                               *
*****************************************************************

HAROSS:PROC(A,N);
     DCL
          (A(N),     /* ADJACENCY MATRIX */
          G,         /*CURRENT VERTEX SET*/
          CLIQ,      /*COMPLETE SUBGRAPH VERTICES */
          GTEMP)
          BIT(N),
          (N,  /* NUMBER OF VERTICES IN GRAPH*/
          WT,  /*NUMBER OF VERTICES IN G */
          VTX(N), /* LABELS OF VERTICES OF G*/
          R(N), /*ROW SUMS OF (A**2XA) FOR G*/
          D(N)) /* DEGRES OF VERTICES OF G */
          FIXED BIN,
          VSET BIT(*) CTL;  /* STACK OF SETS */
     DCL CTR FIXED BIN;
     CTR=0;
     G=G|( G);
     PUT SKIP;
     PUT SKIP LIST ('THE CLIQUES ARE:');
START:
/*
     DETERMINE THE VERTICES IN SUBGRAPH G
*/
     IF G='0'B THEN GO TO NEXT;
     GTEMP=GTEMP|( GTEMP);
     WT=0;
     DO I=1 TO N;
     IF SUBSTR(G,I,1)='0'B THEN GO TO LP1;
     GTEMP=GTEMP&A(I);
     WT=WT+1;
     VTX(WT)=I;
LP1: END;
     IF GTEMP  = '0'B THEN GO TO NEXT;
/*
     CALCULATE ROW SUMS OF (A**2XA)
     AND DEGREES OF VERTICES OF G
*/
     R,D =0;
     DO I=1 TO WT-1;
     DO J=I+1 TO WT;
     SUM=0;
     IF SUBSTR(A(VTX(I)),VTX(J),1)='1'B THEN
          BEGIN;
          D(I)=D(I)+1; D(J)=D(J)+1;
          DO K=1 TO WT;
          IF SUBSTR(A(VTX(I))&A(VTX(J)),VTX(K),1)
               ='1'B THEN SUM=SUM+1;
          END;
          END;
```

```
      R(I)=R(I)+SUM; R(J)=R(J)+SUM;
      END;
      END;
/*
      SEARCH FOR UNICLIQUAL VERTICES
*/
      MIN=1;
      DO I=1 TO WT;
      IF R(I)=D(I)*(D(I)-1) THEN
           DO;
/*
           UNICLIQUAL VERTEX DISCOVERED
*/
           CLIQ=A(VTX(I))&G;
           SUBSTR(CLIQ,VTX(I),1)='1'B;
/*
            IS THIS A MAXIMAL COMPLETE SUBGRAPH?
*/
      GTEMP=GTEMP|( GTEMP);
           DO I2=1 TO WT;
      IF SUBSTR(CLIQ,VTX(I2),1)='1'B THEN
           GTEMP=GTEMP&A(VTX(I2));
           END;
      IF GTEMP='0'B THEN
           PUT LIST(CLIQ);
/*
      DELETE ALL UNICLIQUAL VERTICES IN THIS
      COMPLETE SUBGRAPH FROM G.
*/
           SUBSTR(G,VTX(I),1)='0'B;
           DO J=I+1 TO WT;
           IF SUBSTR(A(VTX(I)),VTX(J),1)='0'B THEN
                GO TO LP2;
           IF R(J)=R(I) THEN SUBSTR(G,VTX(J),1)='0'B;
LP2:       END;
      GO TO START;
           END;
      ELSE
           IF R(MIN)>R(I) THEN MIN=I;
ILP:       END;   /* END I LOOP  */
/*
           NO UNICLIQUAL VERTEX IN G
      DEFINE TWO VERTEX SETS.  SAVE ONE
      AND PROCESS THE OTHER.
*/
      ALLOCATE VSET BIT(N);
      GTEMP=G&A(VTX(MIN));
      SUBSTR(GTEMP,VTX(MIN),1)='1'B;
      VSET= GTEMP;
      DO J=1 TO WT;
      IF SUBSTR(GTEMP,VTX(J),1)='0'B THEN
           VSET=VSET|A(VTX(J));
      END;
      VSET=VSET&G;
      CTR=CTR+1;
      G=GTEMP;
```

```
     GO TO START;
/*
     GET A NEW SET FROM THE STACK.
*/
NEXT:
     IF CTR > 0 THEN
          DO;
          G=VSET;
          CTR=CTR-1;
          FREE VSET;
          GO TO START;
          END;
     FREE VSET;
     RETURN;
     END;   /*  END HAROSS PROCEDURE   */
```

```
*****************************************************************
*                                                               *
*                          BONNER'S ALGORITHM                   *
*                                                               *
*                                                               *
*****************************************************************

BON:PROC(G,N);
     DCL (G(N),A(N),C(N),U,T) BIT(N),L(N);
STEP1: I=1; C(1)='0'B; C(1)=  C(1); A(1)='0'B; L(1)=1;
STEP2: IF SUBSTR(C(I),L(I),1)='1'B THEN
     BEGIN;
STEP3:      C(I+1)=C(I)&G(L(I));
            SUBSTR(C(I+1),L(I),1)='0'B;
            A(I+1)=A(I);
            SUBSTR(A(I+1),L(I),1)='1'B;
STEP4:      L(I+1)=L(I)+1;
            I=I+1;
     END;
     ELSE L(I)=L(I)+1;
STEP5: IF SUBSTR(C(I),L(I),N+1-L(I))  ='0'B THEN GO TO STEP2;
       T=A(I);
STEP6: IF C(I)='0'B THEN
STEP7: I=I-1;  IF I=0 THEN RETURN;
STEP8: U='0'B; SUBSTR(U,L(I)+1,N-L(I))
                          =SUBSTR(C(I),L(I)+1,N-L(I));
     IF (T | U) = T THEN GO TO STEP7;
     L(I)=L(I)+1;
     GO TO STEP2;
        END;   /*END BON PROC  */
```

```
*****************************************************************
*                                                               *
*                  THE MODIFIED PEAY ALGORITHM                  *
*                                                               *
*                                                               *
*****************************************************************

PEAY: PROC(A,N);
     DCL
          (A(N),     /* ADJACENCY MATRIX */
          G,         /* CURRENT VERTEX SET */
          GTEMP)
          BIT(N),
          (N,  /*NUMBER OF VERTICES IN GRAPH*/
          WT,  /* NUMBER OF VERTICES IN G */
          VTX(N))   /*LABELS OF G'S VERTICES*/
          FIXED BIN,
          1 VSTORE BASED(VPTR), /* STACK OF SETS*/
               2 VNXT PTR,
          2 VCTR FIXED BIN,
               2 VN FIXED BIN,
               2 VSET BIT(N REFER(VN)),
          VHD PTR;  /*POINTS TO STACK TOP*/
     DCL P PTR;
     DCL CTR FIXED BIN;
     G=G|( G);
     VHD=NULL;
     CTR=0;
     PUT SKIP;
     PUT SKIP LIST('THE CLIQUES ARE:');
START:WT=0;
/*
     DETERMINE THE LABELS OF THE VERTICES IN G
*/
     DO I=1 TO N;
     IF SUBSTR(G,I,1)='0'B THEN GO TO LP1;
     WT=WT+1;
     VTX(WT)=I;
LP1: END;
/*
     FIND TWO NON-ADJACENT VERTICES IN G
*/
     DO I=1 TO WT;
     GTEMP=A(VTX(I))&G;
     SUBSTR(GTEMP,VTX(I),1)='1'B;
     IF GTEMP  = G THEN GO TO FOUND;
     END;
/*
     G IS A COMPLETE SUBGRAPH.  DETERMINE
     WHETHER IT IS MAXIMAL.
*/
     GTEMP=GTEMP|( GTEMP);
     DO I=1 TO WT;
     GTEMP=GTEMP&A(VTX(I));
     END;
     IF GTEMP='0'B THEN
```

```
/*
     IF STACK NOT EMPTY THEN CHOOSE ANOTHER
     VERTEX SET FOR FURTHER PROCESSING
*/
     IF VHD  = NULL THEN
          DO;
          G=VHD-> VSET;
          P=VHD;
          VHD=VHD->VNXT;
          CTR=CTR-1;
          FREE P->VSTORE;
          GO TO START;
          END;
     RETURN;
FOUND:
/*
     DETERMINE TWO NEW VERTEX SUBSETS,SAVE ONE
     AND PROCESS THE OTHER.
*/
     GTEMP= A(VTX(I));
     DO K=1 TO N;
     IF K  = VTX(I) THEN
     IF SUBSTR(A(VTX(I)),K,1)='0'B THEN
          GTEMP=GTEMP|A(VTX(I));
     END;
     SUBSTR(GTEMP,VTX(I),1)='0'B;
/*
     CHECK THE STACK TO SEE IF NEW SET "GTEMP"
     IS CONTAINED IN A PREVIOUS ONE AWAITING
     PROCESSING.
*/
     P=VHD;
     DO WHILE(P =NULL);
     IF (GTEMP|P->VSET) = P->VSET THEN
          GO TO START;
      P=P->VNXT;
     END;
     CTR=CTR+1;
     ALLOCATE VSTORE;
     VSET=GTEMP&G;
     VNXT=VHD;
     VCTR=CTR;
     VHD=VPTR;
     G=A(VTX(I))&G;
     SUBSTR(G,VTX(I),1)='1'B;
     GO TO START;
     END;   /* END PEAY PROCEDURE  */
```

```
*****************************************************************
*                                                               *
*                  REDUCED REDUNDANCY ALGORITHM                 *
*                                                               *
*                                                               *
*****************************************************************

ENUM:PROC(A,N);
     DCL
          (A(N),     /* ADJACENCY MATRIX */
          G,         /* CURRENT VERTEX SET */
          H,         /* NEWLY DEFINED SET */
          CSUB,      /*COMPLETE SUBGRAPH TO BE
                       EXTENDED BY VERTICES IN G */
          CLQ,       /* NEWLY EXTENDED CSUB */
          F,         /*SET OF DEFINING VERTICES */
          GMX,
          CMX)
          BIT(N),
          (V,        /* VERTEX CHOSEN FROM F */
          N)         /* NUMBER OF VERTICES OF GRAPH*/
          FIXED BIN;
     DCL CTR FIXED BIN;
     DCL VTEMP PTR;
     DCL VSET BIT(*) CTL;  /* STACK OF SETS */
     CTR=0;
     NN=N*2;
     G=G|( G);
     CSUB='0'B;
/*
     DETERMINE WHETHER THERE IS A VERTEX
     ADJACENT TO ALL VERTICES IN G|CSUB. IF
     YES THEN NO CLIQUE IS DEFINED ON G|CSUB
     SO CHOOSE A NEW VERTEX SET.
*/
START:
     GMX=G|CSUB;
       NV=0;
     DO I=1 TO N;
     IF (A(I)|GMX) = A(I) THEN GO TO NEWSET;
     END;
     V=INDEX(G,'1'B);
/*
   F IS THE SET OF VERTICES IN G NOT ADJACENT
     TO V.
*/
     F=G&( A(V));
LOOP:
     H=G&A(V);
     CLQ=CSUB;
   SUBSTR(CLQ,V,1)='1'B;
     IF H='0'B THEN
/*
          NO FURTHER VERTICES CAN BE ADDED TO
          CLQ, HENCE CLQ IS A COMPLETE SUBGRAPH.
          DETERMINE IF IT IS MAXIMAL.
```

```
*/
          DO;
        NV=0;
          DO I=1 TO N;
          IF (A(I)||CLQ) = A(I) THEN GO TO OUT;
          END;
OUT:
          GO TO NXTV;
          END;
/*
     PLACE H AND CLQ ON THE STACK FOR FURTHER
     PROCESSING
*/
     CTR=CTR+1;
     ALLOCATE VSET BIT(NN);
     VSET=H||CLQ;
NXTV:
     SUBSTR(F,V,1)='0'B;
     SUBSTR(G,V,1)='0'B;
     V=INDEX(F,'1'B);
     IF V =0 THEN GO TO LOOP;
/*
     ALL VERTEX SETS HAVE NOW BEEN DETERMINED
     FOR THIS ITERATION.  CHOOSE A NEW SET
     FROM THE STACK FOR FURTHER PROCESSING
*/
NEWSET:
     IF CTR <= 0 THEN RETURN;
     G=SUBSTR(VSET,1,N);
     CSUB=SUBSTR(VSET,N+1,N);
     FREE VSET;
     CTR=CTR-1;
     GO TO START;
     END;  /* END ENUM PROC */
```

```
*****************************************************************
*                                                               *
*     ALGORITHM TO DETECT THE EXISTENCE OF A CLIQUE OF          *
*               ORDER K IN A K-PARTITE GRAPH                    *
*                                                               *
*****************************************************************

KGRPH:PROC(A,N,M,K,MX);
     DCL
          A(N)       /* ADJACENCY MATRIX */
          BIT(N),
          (M(K),     /* NO. VERTICES PER BLOCK */
          MX,        /* MAX. NO. VERTICES/BLOCK */
          N,         /* NO. VERTICES IN GRAPH */
          K,         /* NO. OF BLOCKS */
          R,S,T,VX,VTX,POS,POS2,M1M2) FIXED BIN;
     DCL VB(K) FIXED BIN;
       DCL A0(N) BIT(N);
        DCL U FIXED BIN;
STEP0:A0=A;
     ITER=ITER+1;
     N1=N-M(1)-M(K);
     POS=M(1)+1 ;
     POS2=POS+M(2);
     M1M2=POS2;
     NSP=(N-M1M2)*N1+N-M(K);
BEGIN;
     DCL MSUM(K);
     DCL (P,Q,TEMP,W(NSP,K-2)) BIT(MX);
     W='0'B;
     MSUM(1)=M(1);
     DO I=2 TO K;
     MSUM(I)=MSUM(I-1)+M(I);
     END;
     DO S=1 TO K-2;
     DO I=POS TO N-M(K);
/*
     CHOOSE AN EDGE (I,J) NOT YET DELETED FROM
          FURTHER CONSIDERATION.
*/
     DO J=I+1 TO N;
     IF SUBSTR(A(I),J,1)='1'B THEN
          BEGIN;
          P='0'B;
/*
     MAP THE VERTEX PAIR (I,J) ONTO AN
     INTEGER IJ.   INITIALLY DEFINE THE SET W(IJ,S).
*/
          IJ=(J-M1M2)*N1+I;
          W(IJ,S)=SUBSTR(A(I)&A(J),POS-M(S),M(S));
/*
     DEFINE THE SET W(IJ,T) FOR T=S-1,S-2,...,1
     AS A FUNCTION OF THE PREVIOUSLY DETERMINED
     SETS W(IJ,T+1),W(IJ,T+2),...,W(IJ,S).
*/
          DO NT=1 TO S;
```

```
          T=S+1-NT;
          IF T=S THEN GO TO SKP;
         DO R=T+1 TO S;
          VTX=INDEX(W(IJ,R),'1'B);
          DO WHILE (VTX  = 0);
               VX=VTX+MSUM(R-1);
               IF SUBSTR(A(J),VX,1)&SUBSTR(A(I),VX,1)='0'B THEN
                    BEGIN;
                    SUBSTR(W(IJ,R),VTX,1)='0'B;
                    GO TO SKP3;
                    END;
               Q=W((J-M1M2)*N1+VX ,T)&W((I-M1M2)*N1+VX ,T);
/*
     IF THERE EXISTS A VERTEX  VTX NOT ADJACENT
     TO ANY VERTEX IN W(IJ,T) THEN DELETE IT FROM
     FURTHER CONSIDERATION
*/
               IF Q  = '0'B THEN P=P|Q;
               ELSE SUBSTR(W(IJ,R),VTX,1)='0'B;
               IF VTX >= MX THEN GO TO CHK4;
               TEMP='0'B;
          SUBSTR(TEMP,VTX+1,MX+1-VTX)=
               SUBSTR(W(IJ,R),VTX+1,MX+1-VTX);
               VTX=INDEX(TEMP,'1'B);
SKP3:
               END;
CHK4:
          W(IJ,T)=W(IJ,T)&P;
          END;  /* END R LOOP */
/*
     IF W(IJ,T) IS EMPTY FOR ANY T=1,2,...,S
     THEN DELETE EDGE (I,J) FROM FURTHER
     CONSIDERATION
*/
SKP:      IF W(IJ,T)='0'B THEN
               BEGIN;
     SUBSTR(A(I),J,1)='0'B;
     SUBSTR(A(J),I,1)='0'B;
               GO TO NEXT;
               END;
          END;  /* END T LOOP */
          END;  /* END BEGIN BLOCK */
NEXT:END;  /* END J LOOP */
     END;  /* END I LOOP */
          POS=POS2;
          POS2=M(S+2)+POS2;
    END;  /* END S LOOP */
/*
     TEST WHETHER ALL EDGES HAVE BEEN DELETED
    FROM THE SET OF CANDIDATES.  IF S=K-2
     AND NOT ALL EDGES HAVE BEEN ELIMINATED, THEN
     TEST FOR FURTHER ITERATION.
*/
     DO I=N-M(K)-M(K-1)+1 TO N-M(K);
     DO J=N-M(K)+1 TO N;
     IF W((J-M1M2)*N1+I,1)  = '0'B THEN
```

```
                GO TO STEP19;
      END;
      END;
CONDA:PUT SKIP LIST('NO K-CLIQUE');
        RETURN;
STEP19:DO I=1 TO K;
        VB(I)=0;
        DO J=1 TO M(I);
        U=I*MX+J-MX;
        IF A0(U)  = A(U) THEN GO TO STEP0;
        IF A(U)   = '0'B THEN VB(I)=VB(I)+1;
        END;
        END;
CONDB:PUT SKIP LIST('NO CHANGE');
        DO I=1 TO K;
        IF VB(I) > 2 THEN GO TO STEP20;
        END;
      PUT SKIP LIST('K-CLIQUE EXISTS BY THEOREM 4.3');
      RETURN;
STEP20:CALL ENUM(A,N);
      RETURN;
      END;  /*  END KGRPH */
```

APPENDIX C

graph 1

000101001
000001100
000100101
101000101
000000110
110000110
011111000
1011C0000

graph 2

C00101C01
000111100
CC01CC111
111000111
C1CC00110
110000110
011111C00
101100000

graph 3

000111001
000111110
000111111
111000111
111000110
111000111
011111000
011111000
101101000

graph 4

CCC111111
000111111
C00111111
111000111
111C00111
111000111
111111C00
111111000
111111C00

graph 5

000101001000
000001100000
000100101110
101000101101
000000110110
110000110011
011111000100
000011000101
101100000111
001110111C00
C01011001000
000101011000

graph 6

000101001100
C00111100111
000100111011
111CC0111C00
010000110110
110C00110101
011111000110
001111C00111
101100000011
110011110C00
011010111000
011001011C00

graph 7

000111001110
000111110101
C00111111111
111000111110
111000110111
111000111111
011111000011
011111000011
101101000111
111111001000
101111111000
011011111000

graph 8

000111111111
CC0111111111
000111111111
111000111111
111000111111
111C00111111
111111000111
111111000111
111111000111
111111111C00
111111111000
111111111C00

graph 9

```
0000C00001111001 11
C0000101101011110 1
0000C0001111 0CCCC1
0000C00100000 10101
0000000011110111001
0100C0000110100011
0000C0000 1010CCCC1
0101000000111001 11
0110100000000001CCC
1010111000001111111
1110110100000101CC
1010001100000001000
1100110101 0C0CC010
0101100001100000 10
0100100011010 00C01
1101000101100000 00
1000010101001 1CCCC
1111111101000010 00
```

graph 10

```
C00C01111100110110
0000010001110011101
C00C01C01101100100
0000001110001001 11
CCCCCC01001C101110
1110000000011100011
1001 0CCC0001111011
1001100000110111 10
1011CCCCCCC00110000
1110000000000111011
0100110 1CCCC010110
0110011100000000001
1011111011CC000110
1000001111110000001
0100101101CCC00CC0
1111100100101 00C00
1001111110110100000
0101011001010 10000
```

graph 11

```
0001011011110C111C
0000111111111111111
0000011101111111111
1000001111110100111
0100C001111110 1111
1110C0001111111011
1111000001110111CC
0111100000011011 10
1101110000010010 11
1111111000001111111
1111111C0000000110
1110111110000011111
0111110101CCCC0111
0110011001000000011
1110111111010CCCCCC
1111101101111000000
1111110111111 10CCC0
0111110011011100000
```

graph 12

```
CC01111111111111111
0001111111111111111
CC01111111111111111
1110001111111111111
111CC01111111111111
1110001111111111111
1111110011111111111
1111110001111111111
1111111110001111111
1111111110011111111
1111111111000111111
1111111111111000111
1111111111111000111
1111111111111000111
1111111111111111C00
1111111111111111000
1111111111111111000
```

graph 13

```
0000000001111001110 10
0000110101111101110C01
CCCC0010000010101000 00
0000001011001110111 00
0100000001101000110 01
0110000000C0001001110
C001CC0001000001000 00
0100000000111100101 00
C0010000000100000010 1
1101101000000011010 10
1100100100000011100 00
1110000110000011C1CC1
1101100100000000011 00
0011010100CCC00CCCC11
0101000001110000010 01
1110001001110CCCCCCC0
1101110100101C0000001
1001110001011010CCC00
0001010110001000000 00
1000000001000 10CCCCC0
0100100010010110 10000
```

graph 14

```
000 11111100 10 10 111C00
C0001110001111001111 0
00000 10110111110110110
1CCCCC0011111101101 1 1
1100000 1000 1001101001
111CCC0000011C01011001
1100000000001100111 10
10111CC000111101110 00
10110000000001111111 10
C001CCC00C0010110010 0
0 1110 10100000110010 11
1111110100C0C001010100
0 110001111100000111101
1111001110100000111 10
000 11100111100000C001
10101001110010000011 0
1111011110011100000 10
1101111111010110000001
0 110001011010 111010CC00
0 1110010101001011 0000
000 1110000 10 10 1001C00
```

graph 15

```
0001111111111110C11C11
0000111111111011010101
0000011111111110111101
1000001111010011110 11
1100000101111101110111
1110C0001111111111110
1111C0000010111111C111
1111100000111111111 10
1111010000001101111111
1111110000000111110111
1110111100000111110111
1111110100000000101111
1010111111 10C0C0111111
1110011111110000011111
0101111101100C00001111
0011111111111100000111
1111111111101100CC0011
1011010110011110000001
0110111111111111CC000
1001111111111111110000
1111101011111111111000
```

graph 16

```
0C011111111111111111 1
0000 1111111111111111 1
CCCC011111111111111111
100000 111111111111111
11CCCC0111111111111111
11100000 111111111111
1111CCC001111111111111
1111100000 11111111111
111111C0C001111111111
1111111000001111111111
1111111 10C0001111111 1
1111111110000011111111
1111111111 10C0C0111111
1111111111111000011111
1111111111111C000011 1 1
1111111111111100000111
1111111111111110C00011
1111111111111111CC0C01
1111111111111111100000
1111111111111111110CC0
1111111111111111111000
```