UNIVERSITY
OF TORONTO

AN ALGORITHM TO DETERMINE THE CHROMATIC NUMBER OF A GRAPH

by

Barry Graham

THESIS

Technical Report No. 47, November 1972

DEPARTMENT
OF
COMPUTER SCIENCE

AN ALGORITHM TO DETERMINE THE CHROMATIC NUMBER OF A GRAPH

by

Barry Graham

Technical Report No. 47, November 1972

1A

A Master's Thesis

submitted in accordance with the regulations of the

Department of Computer Science

University of Toronto

October 1972

ABSTRACT

A heuristic algorithm for the determination of the chromatic number of a finite graph is presented. This algorithm is based on Zykov's theorem for chromatic polynomials and extensive empirical tests show that it is the best algorithm available. Christofides' algorithm for the determination of chromatic number is described and is used in the comparison tests.

An Algol-W coding of both algorithms is included in the appendix.

ACKNOWLEDGEMENTS

I wish to thank my advisor, Professor Derek G. Corneil, for his thoughtful advice and valuable guidance throughout the duration of this work. I am also grateful to Professor A. Borodin for his constructive criticism.

Financial assistance was gratefully received from the Department of Computer Science.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER I INTRODUCTION

I.1 Statement of Problem

This thesis presents an algorithm* for the determination of the chromatic number of a graph. Empirical evidence indicates that this algorithm is the best available.

The chromatic number problem belongs to the Cook-Karp class [10] which also includes the subgraph isomorphism, the Hamiltonian circuit and the maximal clique problems. Problems in the Cook-Karp class are polynomially equivalent, that is to say "either each of them possesses a polynomially bounded algorithm or none of them does". A polynomial algorithm (i.e. a procedure which guarantees a solution in a time which is bounded by a polynomial function of the size of the graph) has not been found for any of these problems and Karp [10] suggests that there is strong evidence (but as yet no proof) to support the view that none will be found. Thus in developing a new algorithm the aim is to produce one which has a faster execution time or which requires less storage than existing algorithms. The algorithm presented in this thesis is based on Zykov's theorem for chromatic polynomials.

We now introduce some terminology which is relevant to the chromatic number problem.

* This algorithm is referred to as the Corneil-Graham algorithm.

Definition I.1 Colourings and Chromatic Number.

Let G(V,E) be a graph of order n with the set V of vertices and the set E of edges. A colouring of G is an assignment of a colour to each vertex such that no two adjacent vertices have the same colour. Figure I.1 shows a colouring of a 4-cycle using the three colours A, B and C.

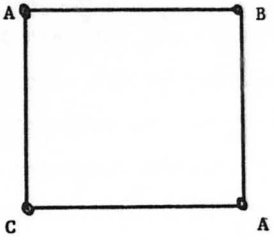


Figure I.1

A colouring is equivalent to a partitioning of the set of vertices of the graph such that adjacent vertices do not belong to the same cell; the vertices in a single cell constitute a colour-class.

A colouring which uses r colours is called an r-colouring and a graph for which an r-colouring exists is said to be r-colourable or r-chromatic.

A colouring is minimal if the graph cannot be coloured using fewer colours. The colouring of Figure I.1 is not minimal since it is possible to colour this graph with two

colours as in Figure I.2.

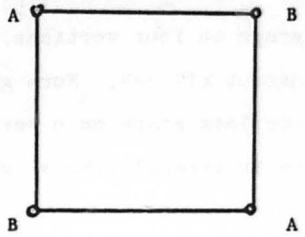


Figure I.2

The chromatic number of G (written $\chi(G)$) is the number of colours used in a minimal colouring of G. The chromatic number of a 4-cycle (Fig. I.2) is two; some further examples follow:

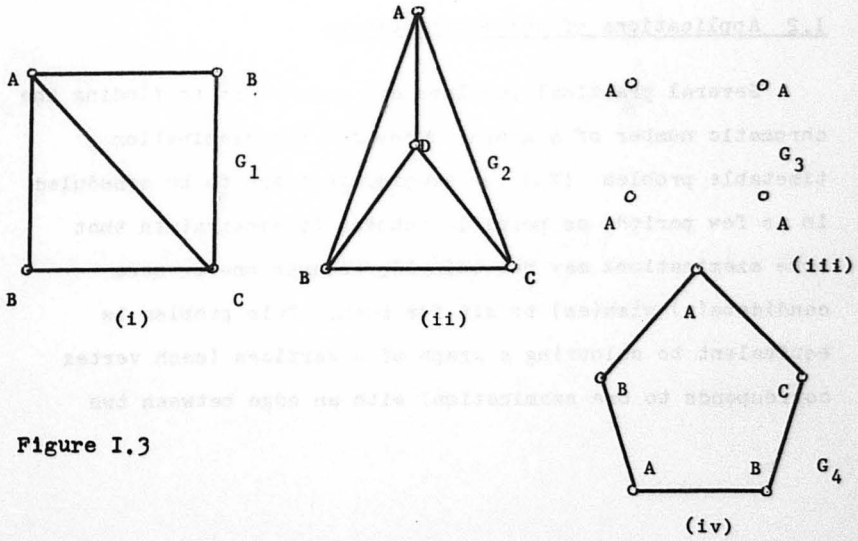


Figure I.3

The colouring in Fig. I.3(i) is minimal since the vertices labelled A, B, and C are mutually adjacent and so must be assigned different colours. Thus $\chi(G_1)=3$.

G_2 is the complete graph on four vertices. Since all vertices are mutually adjacent $\chi(G_2)=4$. More generally $\chi(K_n)=n$ where K_n is the complete graph on n vertices.

Obviously $\chi(G_3)=1$ and in general $\chi(\overline{K}_n)=1$ where $\overline{K}_n$ is the complement of K_n.

$\chi(G_1)=3$ and G_1 contains a 3-clique; $\chi(G_2)=4$ and G_2 is a 4-clique. However, it is not true that a graph with chromatic number k must contain a k-clique. For example G_4 in Fig. I.3(iv) has chromatic number 3 and yet contains no 3-clique. In fact Descartes [5] describes a family of graphs which have arbitrarily high chromatic number and yet contain no triangles.

I.2 Applications of Chromatic Numbers

Several practical problems are equivalent to finding the chromatic number of a graph. Consider the examination timetable problem [23] : n examinations are to be scheduled in as few periods as possible subject to constraints that some examinations may not coincide because one or more candidate(s) wish(es) to sit for both. This problem is equivalent to colouring a graph of n vertices (each vertex corresponds to one examination) with an edge between two

vertices iff the corresponding examinations may not be held at the same time. The chromatic number of the graph gives the fewest periods required to hold all examinations.

This theory is also applicable to the storage problem [4] where chemicals must be stored in compartments and some chemicals may not, for reasons of safety, be placed in the same compartment. Now a vertex of the graph will represent a chemical and an edge between two vertices indicates non-compatibility of the two chemicals. The chromatic number of this graph is the minimum number of compartments required to store all the chemicals safely.

A third example is the channel allocation problem [16]. A country might have more radio stations than frequency channels available so that it becomes necessary for several stations to transmit at the same frequency. However, it might happen that for reasons of proximity, priority etc. that some groups of stations should have different channels. The analogy between this problem and the previous examples should be obvious.

I.3 Existing Algorithms

The chromatic number of a graph G can be found by the following elementary procedure: for r=1,2,3 ... examine all possible assignments of colours to the vertices of G using r colours. The least value r which yields a proper colouring (i.e. such that no two adjacent vertices have the same colour) is the chromatic number of G. Since r colours

may be assigned to n vertices in r^n different ways the number of cases tested before $\chi(G)$ is found will be approximately $\sum_{r=1}^{\chi(G)} r^n$. Even for moderate values of n and $\chi(G)$ this method is unacceptably slow so other solutions were sought. Existing algorithms are of two types:

(i) algorithms which find an approximation to the chromatic number in polynomial time;

(ii) algorithms which find the exact value of the chromatic number but which cannot be guaranteed to find the solution in polynomial time.

Algorithms of both types are heuristics as defined by Slagle [18]: "A heuristic is a rule of thumb, strategy, method or trick used to improve the efficiency of a system which tries to discover the solutions of complex problems. According to 'Websters New Collegiate Dictionary' (1956), the adjective 'heuristic' means 'serving to discover'".

A heuristic algorithm of type (i) is described by Welsh and Powell [22]. The vertices of G are first listed in decreasing order of degree. The first vertex is assigned colour A; next each vertex is inspected in order and any vertex which is not adjacent to a vertex already assigned colour A is itself coloured A. The first uncoloured vertex is now assigned colour B and the process is repeated until all vertices have been coloured whereupon the number of colours used is taken as an approximation to the chromatic

number of G. There are some simple graphs for which this method gives an inaccurate result. Consider, for example, Figure I.4;

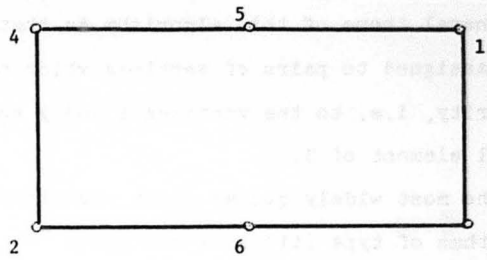


Figure I.4

the vertices are listed and coloured as follows:

vertex	1st scan	2nd scan	3rd scan
1	A		
2	A		
3	/	B	
4	/	B	
5	/	/	C
6	/	/	C

Three colours were used in this colouring of G although it is obvious from inspection that only two colours are necessary.

A heuristic due to Wood [23] is an improvement on the above if the graph has a high density of edges. Wood constructs an n x n similarity matrix $S=\{s_{ij}\}$ where

$$s_{ij} = \begin{cases} 0 \text{ if } (i,j) \text{ is an edge of } G, \\ \#\text{ of vertices adjacent to both } i \text{ and } j \\ \text{otherwise.} \end{cases}$$

The general theme of this algorithm is that colours are first assigned to pairs of vertices which have the highest similarity, i.e. to the vertices x and y such that s_{xy} is a maximal element of S.

The most widely quoted exact algorithms (i.e. heuristic algorithms of type (ii)) are the Zykov [24] and Christofides [4] algorithms. Both these are discussed in some depth in later chapters. In fact the idea for this thesis came from a review of Christofides' algorithm in which Hedetniemi [9] suggests that a comparison of the two algorithms be made.

I.4 Outline of Thesis

In Chapter 2 Zykov's theorem for chromatic polynomials is discussed. A detailed description of the Corneil-Graham algorithm is presented together with some suggestions on how it might be improved. Christofides' algorithm is described in Chapter 3. The empirical tests which were performed to compare the C-G algorithm with that of Christofides, together with the results and conclusions from these tests, are presented in Chapter 4. The appendix contains a listing of the Algol-W coding of the two algorithms.

CHAPTER II THE CORNEIL-GRAHAM ALGORITHM

In this chapter we present the Corneil-Graham (C-G) algorithm for finding the chromatic number of a finite graph. The algorithm is based on Zykov's theorem for chromatic polynomials [24] .

II.1 Chromatic Polynomials

Birkhoff [3] showed that every graph G has an associated polynomial $M_G(\lambda)$, called its chromatic polynomial, which expresses the number of ways that G can be coloured using at most λ colours. For example if $G=K_3$ then

$$M_G(\lambda) = \lambda^3 - 3\lambda^2 + 2\lambda$$

and K_3 can be coloured using at most four colours in $4^3 - 3.4^2 + 2.4 = 24$ ways.

Chromatic polynomials are seen to be relevant to the chromatic number problem when we consider the following lemma:

Lemma 2.1: $\chi(G)$ = least integer i such that $M_G(i) \geq 1$.

The chromatic polynomial for some graphs is readily found. For example consider $\overline{K}_n$.

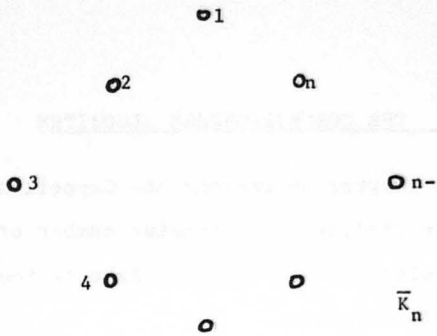


Figure II.1

Since each vertex may be coloured with any of the λ colours we can immediately conclude that

$$M(\lambda) = \lambda^n.$$

Next consider K_n; the first node of this graph can be coloured with any of the λ colours; the second node can be assigned any of the remaining λ-1 colours and so on; thus we can establish that for $G=K_n$

$$M_G = \lambda(\lambda-1)(\lambda-2)\ldots(\lambda-n+1). \qquad \text{(II.1)}$$

For simplicity, following the notation of Read [16] we let $\lambda^{(n)} = M_{K_n}(\lambda) = \lambda(\lambda-1)(\lambda-2)\ldots(\lambda-n+1)$. (II.2)

In order to discuss how to find the chromatic polynomial of an arbitrary graph some definitions are required:

Definition II.1: Let G(V,E) be a graph with non adjacent vertices x and y. We define the <u>reduced graphs of G</u> to be G'_{xy} and G''_{xy} where

(i)$G'_{xy} = G'_{xy}(V',E')$; $V'=V$; $E'=E + (x,y)$;

(i.e. G'_{xy} is obtained from G by adding the edge (x,y).

(ii) $G''_{xy} = G''_{xy}(V'', E'')$; $V''=V-y$;

$(i, j) \in E''$ iff $(i, j) \in E$ for $i, j \neq x$,

$(i, x) \in E''$ iff $(i, x) \in E$ or $(i, y) \in E$;

(i.e. G''_{xy} is obtained from G by identifying (coalescing) the vertices x and y).

For example for G given in Fig. II.2 with x and y

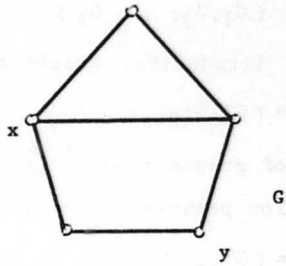


Figure II.2

as shown the reduced graphs are

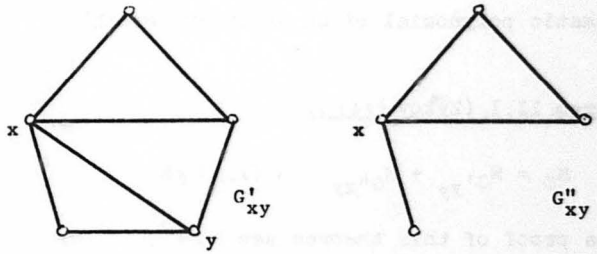


Figure II.3

Obviously any graph which is not complete can be reduced; a complete graph is irreducible. If either G'_{xy} or G''_{xy} is reducible new graphs can be formed by reduction of these; this process can be continued until all graphs obtained are irreducible. At this stage G is said to be completely reduced and

$$R(G) = \{ G_1, G_2, \ldots G_s \} \qquad (II.3)$$

is the set of irreducible graphs thus obtained. Also let

$$R'(G) = \{ G'_1, G'_2 \ldots G'_{s'} \} \qquad (II.4)$$

denote a set of graphs produced at some intermediate stage of the reduction process. For example

$$R'(G) = \{ G'_{xy}, G''_{xy} \}$$

after just one reduction step.

The following theorem provides a method for finding the chromatic polynomial of an arbitrary graph:

Theorem II.1 (Zykov [24]):

$$M_G = M_{G'_{xy}} + M_{G''_{xy}} \qquad \forall (x,y) \notin E .$$

For a proof of this theorem see [8] .

Theorem II.1 can be applied repeatedly to the reduced graphs of G'_{xy} and G''_{xy} and so on until finally we have (cf(II.3) above):

Corollary II.1: $M_G = \sum_{i=1}^{s} M_{G_i}$.

Since each G_i is complete we have M_{G_i} readily available from equation (II.1).

As an example, the chromatic polynomial of a four-cycle is found by reduction as follows;

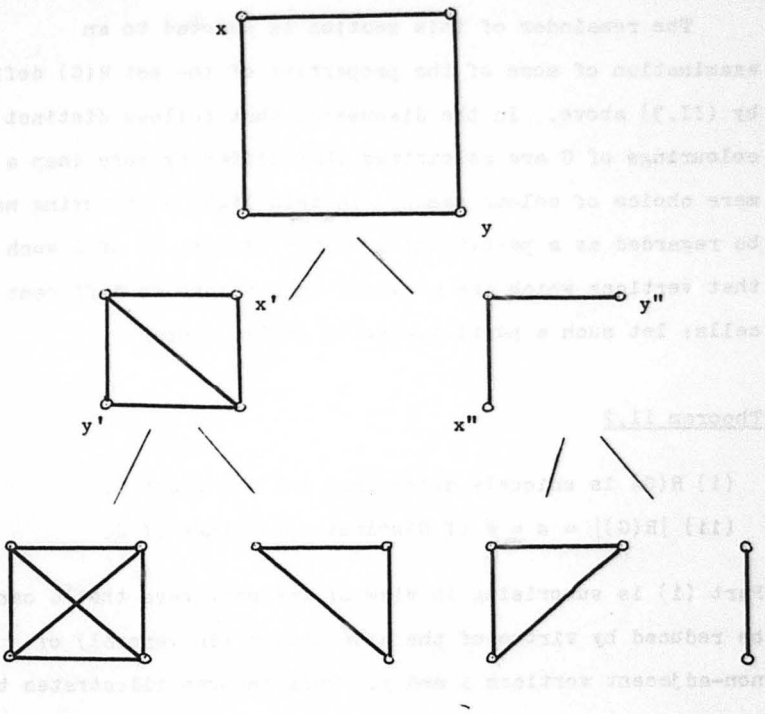


Figure II.4.

Thus $M(\lambda) = \lambda^{(4)} + 2\lambda^{(3)} + \lambda^{(2)}$.

Theorem II.1 together with Lemma II.1 immediately suggests an algorithm for finding $\chi(G)$. The C-G algorithm is based on a variation of Theorem II.1 which is presented as Theorem II.2 below.

The remainder of this section is devoted to an examination of some of the properties of the set R(G) defined by (II.3) above. In the discussion that follows distinct colourings of G are colourings that differ by more than a mere choice of colour names. In this light a colouring may be regarded as a partitioning of the vertex set of G such that vertices which are adjacent in G belong to different cells; let such a partitioning be called proper.

Theorem II.2

(i) R(G) is uniquely determined for any graph G;
(ii) $|R(G)| = s = \#$ of distinct colourings of G.

Part (i) is surprising in view of the many ways that G can be reduced by virtue of the wide choice (in general) of non-adjacent vertices x and y. This theorem illustrates that whatever the intermediate steps in the reduction of G the end result, R(G), will always be the same.

Proof of Theorem II.2.

Definition: Let C(G) be the set of all possible distinct

colourings of G. $C(G) = \{ C_1(G), C_2(G), \ldots C_n(G) \}$ where $C_i(G)$ is the set of all possible distinct i-colourings of G. Thus $C_i(G)$ is the set of all proper partitionings of V which contain i cells and $|C_i(G)|$ is the number of distinct i-colourings of G.

Lemma: If G is reduced to G'_{xy} and G''_{xy} then

$$|C_i(G)| = |C_i(G'_{xy})| + |C_i(G''_{xy})| .$$

Proof: Let $A = \{ c \in C_i(G) \mid [x] \neq [y] \}$

$B = \{ c \in C_i(G) \mid [x] = [y] \}$

Note that $A \cup B = C_i(G)$ and $A \cap B = \phi$.

Now $c \in C_i(G'_{xy}) \Leftrightarrow c \in A$ from Defn. II.1 (i)

therefore $C_i(G'_{xy}) = A$

and $|C_i(G'_{xy})| = |A|$

Also $|C_i(G''_{xy})| = |B|$ from Defn. II.1 (ii)

Since $|A \cup B| = |A| + |B| - |A \cap B|$

then $|C_i(G)| = |C_i(G'_{xy})| + |C_i(G''_{xy})|$. Q.E.D.

This lemma can be applied recursively to the set $R(G) = \{G_1, G_2 \ldots G_s\}$ of graphs obtained when G is completely reduced giving the following result:

$$|C_i(G)| = |C_i(G_1)| + |C_i(G_2)| + \ldots + |C_i(G_s)| \qquad (II.5)$$

Since each $G_j = G_j(V_j, E_j)$ is a complete graph on $|V_j|$

vertices we have

$$|C_i(G_j)| = 1 \text{ if } i = |V_j| \ ;$$
$$= 0 \text{ otherwise.}$$

Thus $|C_i(G)| = \#$ of $K_i \in R(G)$; from (II.5) (II.6)

and $\sum_{i=1}^{n} |C_i(G)| = |C(G)| = s$; from (II.5) (II.7)

Since $|C_i(G)|$ is a function only of G we can conclude from (II.6) that

(i) R(G) is uniquely determined for G

and from (II.7) we have

(ii) $|C(G)| = s = |R(G)|$.

This concludes the proof of Theorem II.2.

II.2 A Simple Algorithm

We now present a known theorem on which the C-G algorithm is based:

Theorem II.3. If x and y are non-adjacent vertices in G then

$$\chi(G) = \min [\chi(G'_{xy}), \chi(G''_{xy})] \ .$$

Proof:

A: If G'_{xy} is r- colourable then trivially there is an r-colouring of G where each vertex of G is coloured as in G'_{xy}. Thus

$$\chi(G) \leq \chi(G'_{xy}) \ . \qquad (i)$$

If G''_{xy} is r-colourable then there is an r-colouring of G where each vertex in G is coloured as in G''_{xy} and vertex y in G is assigned the colour of vertex x. This is possible because every vertex adjacent to either x or y in G is adjacent to x in G''_{xy} and the edge (x,y) is not present in G. Thus

$$\chi(G) \leq \chi(G''_{xy}) \quad . \qquad \text{(ii)}$$

B: If $\chi(G) = r$ then either (a) or (b) (or both) is true:

(a) there exists an r-colouring of G with x and y in different colour classes. Then G'_{xy} is trivially r-colourable and

$$\chi(G) \geq \chi(G'_{xy}) \quad .$$

Thus from (i) we have

$$\chi(G) = \chi(G'_{xy}) \quad .$$

(b) there exists an r-colouring of G with x and y in the same colour class. Then G''_{xy} is r-colourable since every vertex adjacent to x in G''_{xy} is adjacent to either x or y in G.

Thus $\chi(G) \geq \chi(G''_{xy})$

and from (ii) we have

$$\chi(G) = \chi(G''_{xy}).$$

Therefore $\chi(G) = \chi(G'_{xy})$ or $\chi(G) = \chi(G''_{xy})$. (iii)

From (i), (ii) and (iii) it immediately follows that

$$\chi(G) = \text{Min}\,[\chi(G'_{xy}),\ \chi(G''_{xy})]\ .$$

Q.E.D.

Theorem II.3 may be applied recursively so when G is completely reduced to R(G) (c.f. (II.3) we have

$$\chi(G) = \min\,[\chi(G_1),\ \chi(G_2),\ \ldots\ \chi(G_s)]\ . \qquad \text{(II.7)}$$

Similarly when G is partially reduced to R'(G) (c.f. (II.4)) we have

$$\chi(G) = \min\,[\chi(G'_1),\ \chi(G'_2),\ \ldots\ \chi(G'_{s'})]\ . \qquad \text{(II.8)}$$

For example we can see from Fig. II.4 that the chromatic number of a four-cycle is

$$\chi(G) = \min\,[\chi(K_4),\ \chi(K_3),\ \chi(K_3),\ \chi(K_2)]$$
$$= \min\,[\ 4\ ,\ 3\ ,\ 3\ ,\ 2\ ]$$
$$= 2.$$

As a further example we will find the chromatic number of the graph in Fig. II.2. The complete reduction of this graph is shown in Fig. II.5.

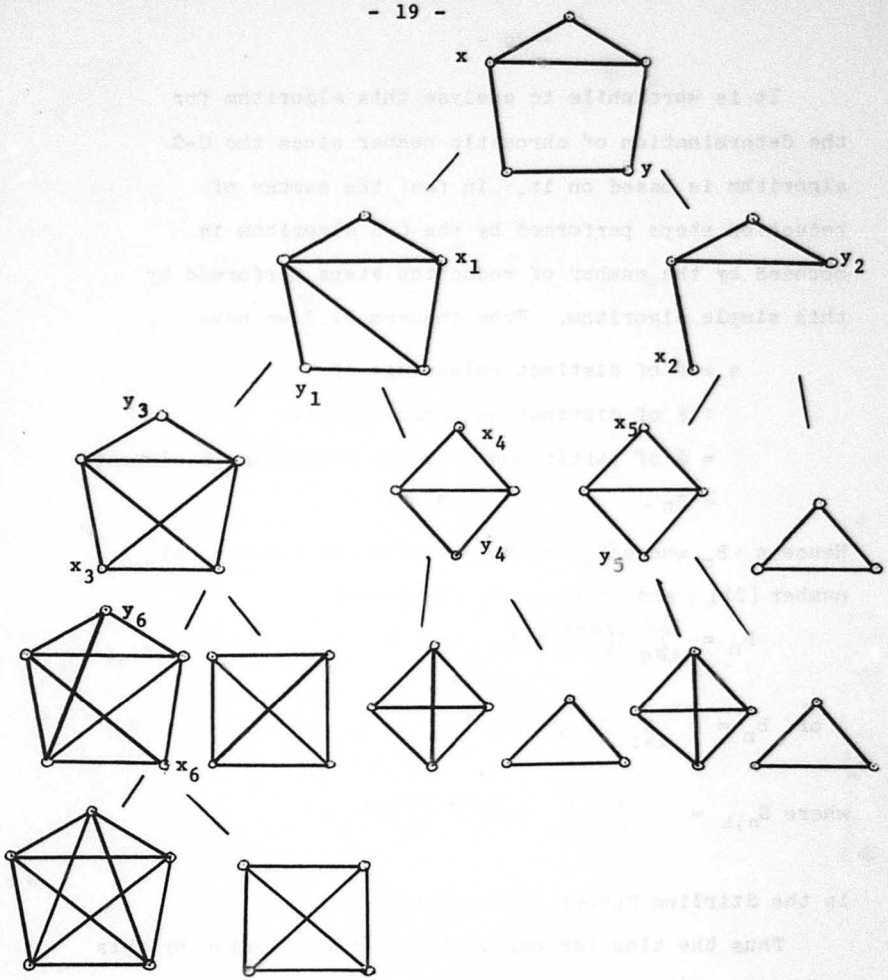


Figure II.5.

Thus $\chi(G) = \min [5,4,4,4,3,4,3,3]$

$= 3.$

It is worthwhile to analyse this algorithm for the determination of chromatic number since the C-G algorithm is based on it. In fact the number of reduction steps performed by the C-G algorithm is bounded by the number of reduction steps performed by this simple algorithm. From theorem II.2 we have

$$\begin{aligned} s &= \#\text{ of distinct colourings of } G \\ &\leq \#\text{ of distinct colourings of } \overline{K}_n \\ &= \#\text{ of partitions of a set containing n elements} \\ &= B_n . \end{aligned}$$

Hence $s \leq B_n$ where B_n is the nth Bell (or exponential) number [21] and is given by the formula

$$B_n = \sum_{k=0}^{n-1} \left[\binom{n-1}{n} B_k\right], \quad B_0 = 1$$

$$\text{or} \quad B_n = \sum_{k=1}^{n} S_{n,k}$$

$$\text{where } S_{n,k} = \frac{1}{n!} \sum_{k=o}^{n} (-1)^{n-k} \binom{n}{k} k^n$$

is the Stirling Number of the second kind.

Thus the time (or magnitude) of the solution by this method is $\leq O(B_n)$. Experimental tests have indicated that $B_n \approx O(2^{n^{1.3}})$ and certainly $O(B_n) << O(2^m) \leq O(2^{n^2})$, which is the bound suggested by Wagner [20], where m is the number of missing edges in G.

II.3 The Improved Algorithm.

In this section we show how the number of reduction steps required by the simple algorithm described in the previous section may be reduced by using a branch and bound approach. Lawlor and Wood [12] describe branching as "an intelligently structured search of the space of all feasible solutions. Most commonly the space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets". In this case a feasible solution is a colouring of the graph and branching is achieved by expressing $\chi(G)$ as the minimum of $\chi(G'_{xy})$ and $\chi(G''_{xy})$ (c.f. theorem II.3); colourings of G'_{xy} are equivalent to colourings of G in which vertices x and y have different colours while colourings of G''_{xy} are equivalent to colourings of G in which x and y have the same colour. Figure II.5 shows how branching propagates for a specific example.

Consider next how the branching may be bounded by considering an intermediate stage in the development of Figure II.5.

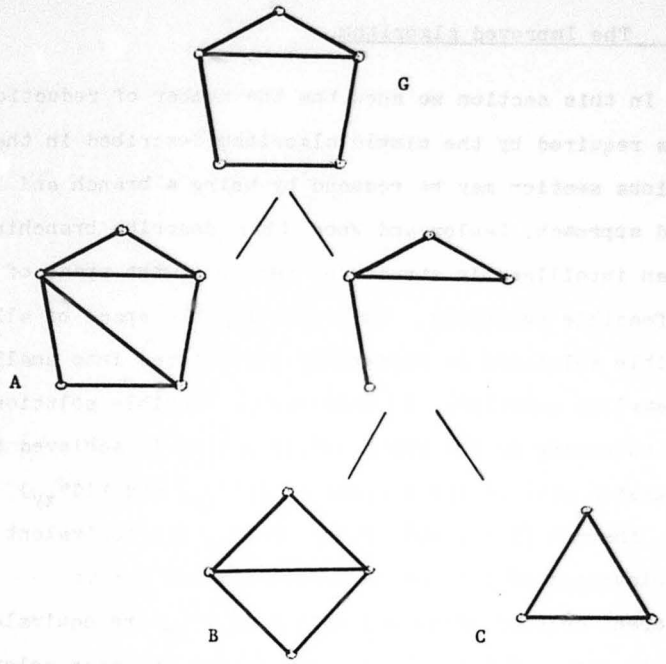


Figure II.6

At this stage we know from equation (II.8) that

$\chi(G) = \min [\chi(A), \chi(B), \chi(C)]$.

Graph C is complete so $\chi(C) = 3$ and 3 is thus an upper bound for $\chi(G)$; note that each of A and B contains a 3-clique so that 3 is a lower bound for $\chi(A)$ and $\chi(B)$. Consequently branching from A and B is unnecessary; we can conclude that $\chi(G)=3$ without finding $\chi(A)$ or $\chi(B)$.

In general whenever α has been established as an upper bound for $X(G)$ then a graph encountered in the reduction of G which is known to contain an α-clique need not be reduced further. This is the essence of bounding and is a major step in the development of the C-G algorithm from the basic method described in section II.2.

The basic bounding strategy, once α has been determined to be an upper bound for $X(G)$, is now obvious. It would, however, be poor practice to examine a graph G_1 which is a candidate for reduction to determine whether it contains an α-clique since there is no 'good' algorithm for finding cliques [10] . Instead the strategy is to find an α-cluster in G_1 (see clustering algorithm in section II.4) where an α-cluster is a set of α vertices which has a high density of edges; let $G_{1\alpha}$ denote the subgraph of G_1 determined by this α-cluster. Next G_1 is reduced to G'_1 and G''_1 such that G'_1 is formed by adding an edge to $G_{1\alpha}$. This process is repeated with G'_1 until the α-cluster becomes an α -clique whereupon this branch is terminated. Graphs formed by coalescence (like G''_1) are treated similarly.

If α is the exact value of $X(G)$ then the above procedure will always terminate successfully by building α -cliques. However, if $\alpha > X(G)$ then at some stage a

graph introduced by coalescence will have only α-1 vertices implying that $\chi(G) \leq \alpha$-1. Whenever this occurs the value α-1 is substituted for α and the execution of the algorithm continues uninterrupted (i.e. a change in the value of α does not require a repetition of any of the previous steps); it terminates when every branch introduced by reduction has been forced to contain an α-clique. Upon termination the value of $\chi(G)$ will be exactly α .

To eliminate any confusion we present a concise recursive definition of the algorithm; its most important feature is the procedure REDUCE (which might be more aptly labelled REDUCE IF NECESSARY) which has two parameters: G, a graph and N, the order of G. Note that the algorithm essentially performs a pre-order traversal [11] of a binary tree such as Fig. II.6 where each node of the binary tree is a graph. In the pre-order traversal the nodes of the tree are visted as follows:

(i) the root is visited;

(ii) the right sub-tree is visited;

(iii) the left sub-tree is visited.

Recursive Definition of the C-G Algorithm.

Main Program:

Find Initial α , an upper bound for $\chi(G)$ (see section II.4)

Reduce (G,n);

Stop (now $\chi(G) = \alpha$).

Procedure Reduce (G,N)

If $\alpha > N$ then $\alpha := N$;

$\beta := \alpha$; (β is a local variable)

If G is complete GO TO EXIT;

Find G_α, a cluster on α-vertices (see section II.4)

A: If G_α is an α-clique GO TO EXIT;

Choose non-adjacent vertices x and y in G;

Form G'_{xy} and G''_{xy} by reduction of G.

$G := G'_{xy}$;

Reduce (G''_{xy}, N-1);

If $\beta = \alpha$ then GO TO A

else Reduce (G,N);

(The 'else' is necessary in case α was changed during the previous step.)

EXIT: END;

The step "choose vertices x and y in G_α " is not precisely defined since, in general, there will be a wide

choice of missing edges in G_α ; at different stages the algorithm employs two contrasting strategies to determine which pair of vertices to choose. During the initial reduction steps it is likely that $\alpha > \chi(G)$ so at this stage we wish to produce by coalescence a graph G''_{xy} which has low chromatic number. In an attempt to achieve this, G''_{xy} should have as few edges as possible; this is tantamount to saying that x and y should have many common adjacent vertices (so that many edges will be coalesced when G''_{xy} is formed). At some later stage it will be quite likely that $\alpha=\chi(G)$ so that the strategy should be to terminate each remaining branch as quickly as possible by forming α-cliques. To build up cliques quickly few edges should be destroyed when G''_{xy} is formed, and so x and y should have few mutually adjacent vertices.

It is not yet determined at what stage in the execution of the reduction process this change of strategy should occur; further tests must be done to "tune" the algorithm.

II.4 Heuristics Employed in the Corneil-Graham Algorithm

In this section we describe three heuristics which are used in the C-G algorithm:

(i) to determine α , an initial approximation and upper bound for $\chi(G)$;

(ii) to find an α-cluster in a graph;

(iii) to remove a vertex from a graph without altering its chromatic number.

(i) To determine α :

The C-G algorithm works fastest when the initial value of α is a close approximation to $\chi(G)$. From Theorem II.3 we know that $\chi(G) = \chi(G'_{xy})$ or $\chi(G''_{xy})$ and there is usually a wide choice for the pair x,y of vertices chosen for the reduction operation. If the choice for x and y satisfies the equality

$$\chi(G) = \chi(G''_{xy}) \qquad (a)$$

we then have a G''_{xy} which has n-1 vertices and the same chromatic number as G. To find $\chi(G''_{xy})$ the above procedure is repeated, perhaps several times, at each step producing a graph with one less vertex than its predecessor and which has chromatic number equal to $\chi(G)$. Of course the process cannot continue indefinitely and eventually an irreducible (complete) graph K_α will be produced. Then

$$\chi(G) = \chi(K_\alpha) = \alpha .$$

This would be an excellent method of calculating $\chi(G)$ except that it is not known how to force $\chi(G''_{xy})$ to equal $\chi(G)$; the heuristic chooses x and y so

that the equality holds in "most" cases. We have already seen that a colouring of G''_{xy} is equivalent to a colouring of G in which vertices x and y have the same colour; so the problem of selecting x and y is equivalent to finding a pair of vertices which are likely to have the same colour in some minimal colouring of G. A good choice is for x and y not to be adjacent and to have several other vertices of G adjacent to both x and y . The same idea is reflected in Wood's algorithm [23] discussed in section I.3. The heuristic may be summarized as follows:

Given G(V,E):

Step 1. If G is complete go to step 6.

Step 2. Set up an nxn similarity matrix $S= \{s_{ij}\}$.

where $$s_{ij}=\begin{cases} 0 \text{ if } (i,j) \in E; \\ \#\text{ of vertices adjacent to both i and j} \\ \text{if } (i,j) \notin E . \end{cases}$$

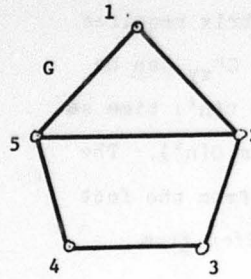
Step 3. Scan S to find a maximal entry s_{xy}.

Step 4. Form G''_{xy} from G by coalescing vertices x and y.

Step 5. Set $G := G''_{xy}$; n:=n-1;
Go to Step 1.

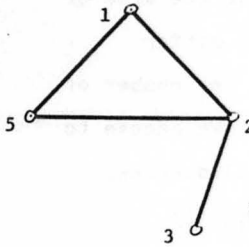
Step 6. $\alpha := n$.
STOP.

Figure II.7 shows how the heuristic is applied in a specific case.



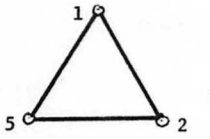
n = 5

$$S = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 & 2 \\ 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix} \qquad x = 4;\ y = 2.$$



n = 4

$$S = \begin{bmatrix} 0 & 0 & 1 & - & 0 \\ 0 & 0 & 0 & - & 0 \\ 1 & 0 & 0 & - & 1 \\ - & - & - & - & - \\ 0 & 0 & 1 & - & 0 \end{bmatrix} \qquad x = 3;\ y = 1.$$



n = 3

Since this graph is complete $\alpha = 3$.

Figure II.7.

Although setting up one similarity matrix requires $O(n^3)$ operations the similarity matrix for G''_{xy} can be deduced from the similarity matrix of G in $O(n^2)$ time so that the overall timing of the heuristic is $O(n^3)$. The usefulness of the heuristic can be gauged from the fact that in none of the cases tested did α differ from $\chi(G)$ by more than two.

(ii) The Clustering Algorithm:

Much research has recently been done in the area of clustering (e.g.[1,13]) but our requirements differ slightly from the usual requirements since the number of vertices, α , has already been determined. We choose to use the threshold matrix method [7] for clustering.

Initially an nxn weight matrix $W = \{w_{ij}\}$ is constructed where the weight entry w_{ij} is somehow a measure of how likely it is that vertices i and j will belong to an α-cluster. The two most important considerations for determining w_{ij} are :

(i) is i adjacent to j in G?

(ii) how many paths of length 2 from i to j exist in G?

The relative importance of (i) and (ii) in forming a cluster was found to vary and it was finally decided that they should be considered equally relevant; a value of $\alpha/2$ is contributed to w_{ij} if (i) is true and 1 is added to w_{ij} for each path of length 2 from i to j up to a

maximum of $\alpha/2$.

Next a threshold value τ is established close to the maximum element of W and a binary threshold matrix $T = \{t_{ij}\}$ is constructed where

$$t_{ij} = \begin{cases} 1 \text{ if } w_{ij} \geq \tau ; \\ 0 \text{ otherwise.} \end{cases}$$

The threshold matrix is interpreted as the adjacency matrix of a graph G_T; because τ is large G_T will, in general, consist of several smaller connected components. If none of these components is of order α or greater then τ is reduced and a new G_T is constructed. This process is continued until at least one of the components of G_T has α or more vertices; at this point a component $C = \{c_1, c_2, \ldots c_r\}$ of order r is chosen such that there is no other component of order r' where $\alpha \leq r' < r$. The final step is to select α of these r vertices to form the cluster; we do this as follows:

1. Initialize the set V_α with the vertex $c \in C$ which has the greatest $\sum_{i=1}^{r} w_{cc_i}$.

2. If $V_\alpha = \{v_1, v_2, \ldots v_s\}$ then add to V_α the vertex $c \in C - V_\alpha$ which has greatest $\sum_{i=1}^{s} w_{cv_i}$.

Repeat step 2 until the cardinality of V_α is α.

This clustering algorithm, which is $O(n^3)$, has been found effective in finding a good α-cluster.

(iii) <u>To remove a vertex from a graph without altering its chromatic number:</u>

Since the clustering algorithm is $O(n^3)$ it would be beneficial to reduce the order of a graph which is about to be reduced provided this can be done without altering its chromatic number. If $Adj(x)$ is the set of vertices adjacent to vertex x in G and there is a vertex y in G such that

$$Adj(x) \subseteq Adj(y)$$

then vertex x and all edges terminating at x can be removed without altering the chromatic number of G. To prove this consider any colouring of G-x; vertex x can be coloured without introducing a new colour by assigning it the same colour as vertex y; this is possible since any vertex which is adjacent to x in G is also adjacent to y. Consequently before a graph introduced during reduction (eg. G") is further reduced a test is applied to determine whether any of its vertices may be eliminated by virtue of this property.

Figure II.8 shows some examples where the above heuristic can be applied to advantage.

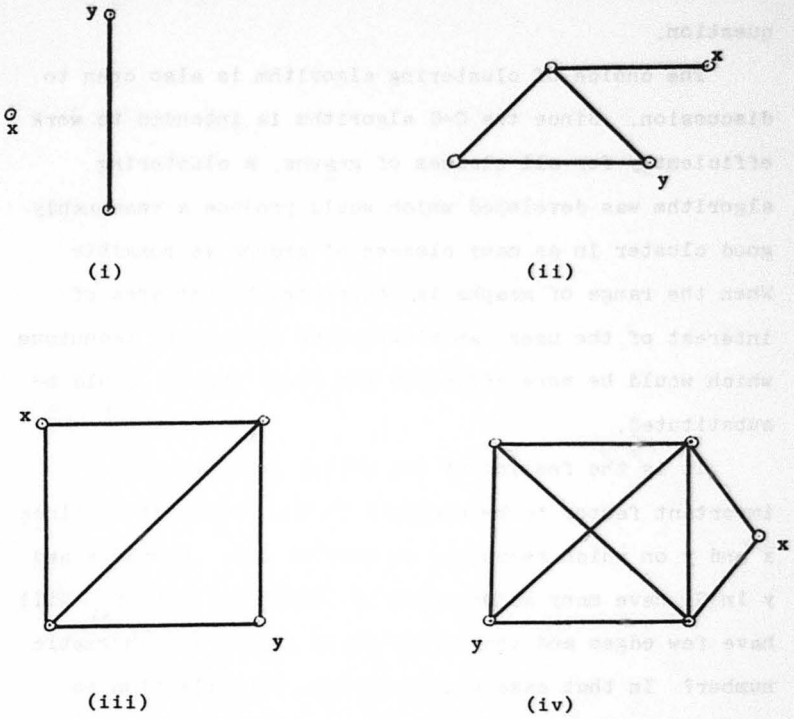


Figure II.8.

II.5 Some User-Adjustments to the Algorithm

As suggested in section II.3 the C-G algorithm is not yet fully 'tuned'. For example the effect of a poor initial approximation to $\chi(G)$ must be investigated; whether it would be worthwhile or not to use a slower

but more accurate heuristic for this is still an open question.

The choice of clustering algorithm is also open to discussion. Since the C-G algorithm is intended to work efficiently for all classes of graphs, a clustering algorithm was developed which would produce a reasonably good cluster in as many classes of graphs as possible. When the range of graphs is restricted by the area of interest of the user, an alternative clustering technique which would be more efficient for these graphs, could be substituted.

It is the feeling of the author that the most important factor to be analyzed is the choice of vertices x and y on which reduction is carried out. Should x and y in G_i have many adjacencies in common so that $G''_{i_{xy}}$ will have few edges and thus most likely have a low chromatic number? In that case a poor initial approximation to X(G) would be more quickly corrected but at the expense of taking many steps to build up α-cliques. Choosing x and y to have few common adjacencies has the opposite effect. Again due to the wide range of graphs being tested, an intuitive and somewhat arbitrary compromise was necessary. Undoubtedly a strategy which reflects some property of the graphs being tested is possible and would lead to a faster execution time. An example of such a modified strategy is described in Section IV.2(iv).

CHAPTER III CHRISTOFIDES' ALGORITHM

This chapter describes Christofides' method as originally conceived by Christofides [4] and as interpreted later by Furtado et al [6] .

III.1 Maximal Internally Stable Sets

The algorithm depends strongly on the concept of a Maximal Internally Stable Set (or MISS) as defined by Berge [2].

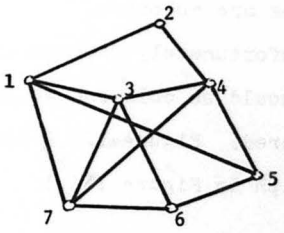
A MISS of a graph is a set of vertices no two of which are adjacent and which is maximal with respect to this property. For example the MISS of Figure III.1 are

$M_1 = \{1,4,6\}$

$M_2 = \{2,3,5\}$

$M_3 = \{2,5,7\}$

$M_4 = \{2,6\}$



FigureIII.1.

For some families of graphs, m, the number of MISS in the graph grows exponentially with the size of the graph. For example a graph consisting of t disjoint triangles will have 3^t MISS; the graph on 24 vertices

consisting of 8 disjoint triangles has $3^8 = 6561$ MISS.

A MISS of G is a clique in $\bar{G}$ (the compliment of G) and v.v. Thus any clique-finding algorithm applied to $\bar{G}$ will find all the MISS of G. Tests have shown [15] that the best clique (and consequently MISS) algorithm is the Bron-Kerbosch algorithm [15].

III.2 Description of Christofides' Algorithm

Having found all the MISS of the graph, Theorem III.1 is used to find the chromatic number.

Theorem III.1 [4] : If a graph G is r-chromatic then it can be coloured with r-colours, colouring first with one colour a MISS of G, say M_1, next colouring with another colour a MISS of $G-M_1$ and so on until all the vertices are coloured.

Unfortunately Theorem III.1 does not determine which MISS should be coloured at each step so every MISS must be considered. Figure III.2 shows how the algorithm colours the graph in Figure III.1.

Colour A	Uncoloured Vertices	Colour B	Uncoloured Vertices	Colour C
$M_1=\{1,4,6\}$	2,3,5,7	$M_{11}=\{2,3,5\}$	7	$M_{111}=\{7\}$
		$M_{12}=\{2,5,7\}$	3	all vertices are
$M_2=\{2,3,5\}$	1,4,6,7	$M_{21}=\{1,4,6\}$	7	now coloured so
		$M_{22}=\{7\}$	1,4,6	STOP; $\chi(G) = 3$
$M_3=\{2,5,7\}$	1,3,4,6	$M_{31}=\{1,4,6\}$	3	
		$M_{32}=\{3\}$	1,4,6	
$M_4=\{2,6\}$	1,3,4,5,7	$M_{41}=\{1,4\}$	3,5,7	
		$M_{42}=\{3,5\}$	1,4,7	
		$M_{43}=\{5,7\}$	1,3,4	

Figure III.2

III.3 Improvements to the Algorithm

A modification introduced by Furtado [6], while not eliminating the exponential nature of the algorithm, reduces the number of MISS investigated; whenever a new colour is introduced Furtado concentrates on colouring only one of the uncoloured vertices and only the MISS which contain this vertex are considered. For obvious reasons the vertex which appears in fewest MISS is selected. With this modification the colouring of Fig. III.1 would be executed as follows:

MISS	Col.A	Uncol. Verts.	MISS	Col.B	Uncol. Verts.	MISS	Col.C
{146} {235} {257} {26}	1,4,6	2,3,5,7	{235} {7}	2,3,5	7	{7}	7
Vertex 1 chosen.			Vertex 2 chosen.			Vertex 7 chosen.	

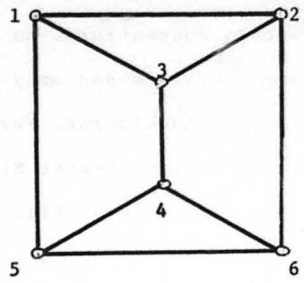
Figure III.3.

It is fortunate in this example that each time a vertex is chosen it appears in only one MISS.

Furtado also suggests that it is necessary to use the MISS (or clique) algorithm only once since the MISS of G can be used to find the MISS of $G-M_i$. This is done by first deleting from each MISS of G any vertex contained in M_i. From this set of 'modified MISS' remove any 'MISS' which is a proper subset of another; the remainder is the

set of MISS of $G-M_i$.

Figure III.4 shows a further example using the modified Christofides' algorithm.



MISS	Col.A	Uncol. Verts.	MISS	Col.B	Uncol. Verts.	MISS	Col.C
{36}			{25}				
{16}	1,6	2,3,4,5	{35}	3,5	2,4	{24}	2,4
{25}			{24}				
			Vertex 3 chosen			Vertex 2 chosen	
{35}			{36}				
{14}	1,4	2,3,5,6	{25}	2,5	3,6	{36}	
{24}			{35}				
Vertex 1 chosen			Vertex 2 chosen				

Figure III.4.

For a formal description of the modified Christofides algorithm see [6].

CHAPTER IV TESTS AND RESULTS

In this chapter an empirical comparison of the C-G and Christofides' algorithms is presented. Four families of graphs used in the tests are described and a detailed analysis of the test results is given.

IV.1 Testing Techniques

The C-G and Christofides algorithms were programmed by the author and run on an IBM 370 model 165 computer. At present the programmes will accept graphs with a maximum of n=64 vertices. In order to minimize storage requirements, binary arrays were packed such that one bit of core represented an element of the array. The execution time of a single graph was limited to five minutes.

Time Measurement

Execution time of a section of programme was determined from successive readings of the built-in computer clock. This clock gives the C.P.U. time elapsed during execution of the programme exclusive of any wait or idle time. It gives readings accurate to 1/60 sec.

Storage Measurement

The storage allocated to a programme can be estimated

from the listings given in the appendix. For this discussion let the unit of storage be one word (or 32 bits). Since we have $n \leq 64$ it requires two units to retain a quantum of information about each vertex.

In Christofides' algorithm the m MISS (Maximal Internal Stable Sets) are stored in the array MISS requiring 2m units of storage. Consider next the two-dimensional arrays STATE and PRESVERTEX which are used to store the uncoloured MISS and vertices of the graph. In order to minimize R, the range of the first subscript of these arrays, some clever use of dynamic programming is necessary. The range of the second subscript of these arrays is $\lceil \frac{m}{32} \rceil$ and 2 respectively. Therefore the overall storage requirement of Christofides' algorithm is approximately

$$2m + R(\lceil \tfrac{m}{32} \rceil + 2) \text{ units.} \qquad \text{(IV.1)}$$

However the implementation of dynamic storage allocation is expensive with regard to execution time and in order to reduce the execution time of Christofides' algorithm this device was not used; instead a huge static memory space is allocated to these arrays with no overlays. However, for the purpose of comparing the two algorithms, expression (IV.1) was used to estimate the storage required by each graph. In this way Christofides' algorithm was given the advantage of dynamic storage allocation without any increase of execution time.

It was demonstrated in section III.1 that m, the number of MISS in a graph, can grow exponentially with the size of the graph. Thus it is obvious from (IV.1) that the maximum storage requirement of Christofides' algorithm grows exponentially with n.

In the C-G algorithm the procedure REDUCE has two arrays which require 2n and 2 units of storage respectively; the variable MAXDEPTH records the maximum depth of recursive calls to the procedure. Thus the storage allocated to this program is given by

$$\text{MAXDEPTH } (2n + 2). \qquad \text{(IV.2)}$$

MAXDEPTH is bounded above by $n - \chi(G) + 1 \leq n$ so that the maximum storage requirement of the C-G algorithm is $O(n^2)$.

IV.2 Test Graphs and Results

The graphs used to compare the two algorithms form four groups:

(i) Pseudo-random graphs;

(ii) Modified Moon Moser;

(iii) Cycles;

(iv) Starred Polygons.

(i) Pseudo-random graphs.

A pseudo-random graph of order n is easily generated when ρ, the density of edges is specified. The maximum number of edges in a graph of order n is $n(n-1)/2$ and so the actual number of edges is $\rho \cdot n(n-1)/2$. Graphs were tested for $\rho = .3, .4, .6, .8, .9$ and for values of n from 10 increasing in steps of 5 until the C.P.U. time used became unreasonably large. By using different initial random numbers, several graphs with the same n and ρ values can be generated; where execution time permitted five graphs were generated for each (n, ρ) value and the average results are recorded.

Table IV.1 shows the observed time and storage requirements for the two algorithms while Figures IV.1 to IV.5 illustrate these results graphically on a log(time) vs n scale. Note from Table IV.1 that in terms of absolute time the C-G algorithm is in all cases superior. A further important comparison is the slope s of the plots (Figures IV.1 to IV.5) since s, the exponent in the equation $\text{Time} = A.n^s$, determines the behaviour of an algorithm for large values of n. In these plots, note that for low values of ρ the C-G algorithm has the advantage of a smaller slope; as ρ increases, this superiority diminishes until at $\rho = .9$ the slopes are approximately equal.

ρ	n	average times* C-G	average times* Christofides	core used† C-G	core used† Christofides
.3	15	12.3	29.7	83.2	98.2
	20	24.0	407.3	75.6	412.2
	25	323.3	13649.0	468.0	9416.5
	30	518.3	-	868.0	-
	35	2471.7	-	1382.4	-
	40	6990.3	-	1689.2	-
.4	10	3.0	3.3	22.0	29.2
	15	11.7	32.7	57.6	94.2
	20	66.3	608.7	268.8	653.0
	25	354.2	6324.2	702.0	4197.5
	30	972.3	-	830.8	-
	35	12785.0	-	1620.0	-
.6	10	3.3	4.0	30.8	31.2
	15	13.7	21.0	76.8	63.6
	20	87.7	175.0	260.4	262.0
	25	418.0	1477.0	353.6	1278.0
	30	3973.3	-	1066.4	-

Table IV.1. Observed time and storage requirements on pseudo-random graphs.

(continued overleaf)

* 1 time unit = .01 sec.

† 1 unit of core = 1 word = 32 bits.

		average times		core used	
ρ	n	C-G	Christo-fides	C-G	Christo-fides
.8	10	1.3	2.7	26.4	23.4
	15	6.0	11.0	38.4	41.8
	20	16.7	27.3	75.6	61.6
	25	177.7	361.3	291.2	308.0
	30	845.3	4909.7	570.4	3323.2
	35	2006.0	-	950.4	-
	40	7825.0	-	1284.7	-
.9	10	1.3	3.7	22.0	22.4
	15	3.7	9.3	32.0	32.0
	20	9.7	19.3	42.0	45.6
	25	19.3	39.0	62.4	64.6
	30	44.3	73.0	124.0	87.2
	35	116.7	254.3	216.0	156.0
	40	347.7	801.0	360.8	397.2
	45	1996.7	2951.0	883.2	1343.8
	50	2255.8	-	2244.0	-

Table IV.1. (continued) Observed time and storage requirements on pseudo-random graphs.

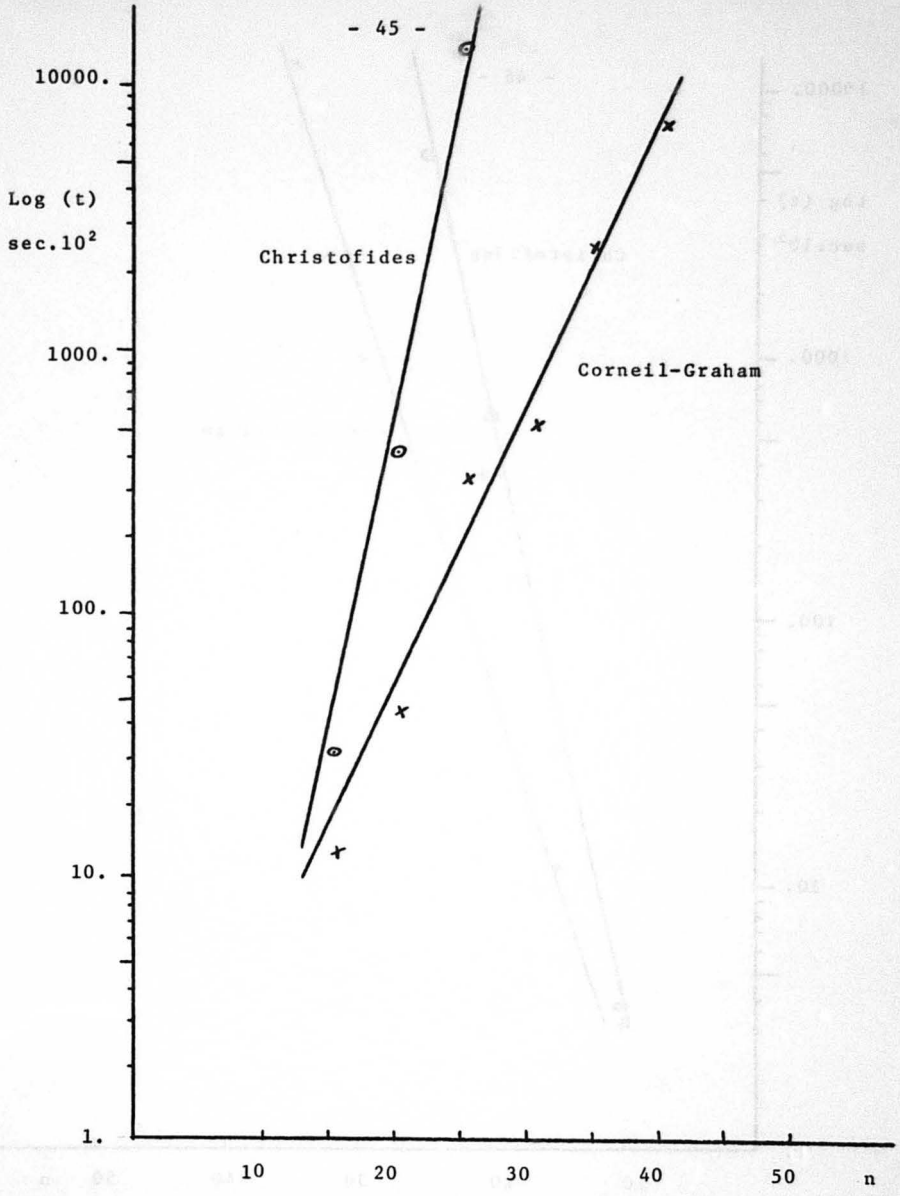


Figure IV.1: Log (time) vs. n. Comparative timings on pseudo-random graphs (ρ=.3).

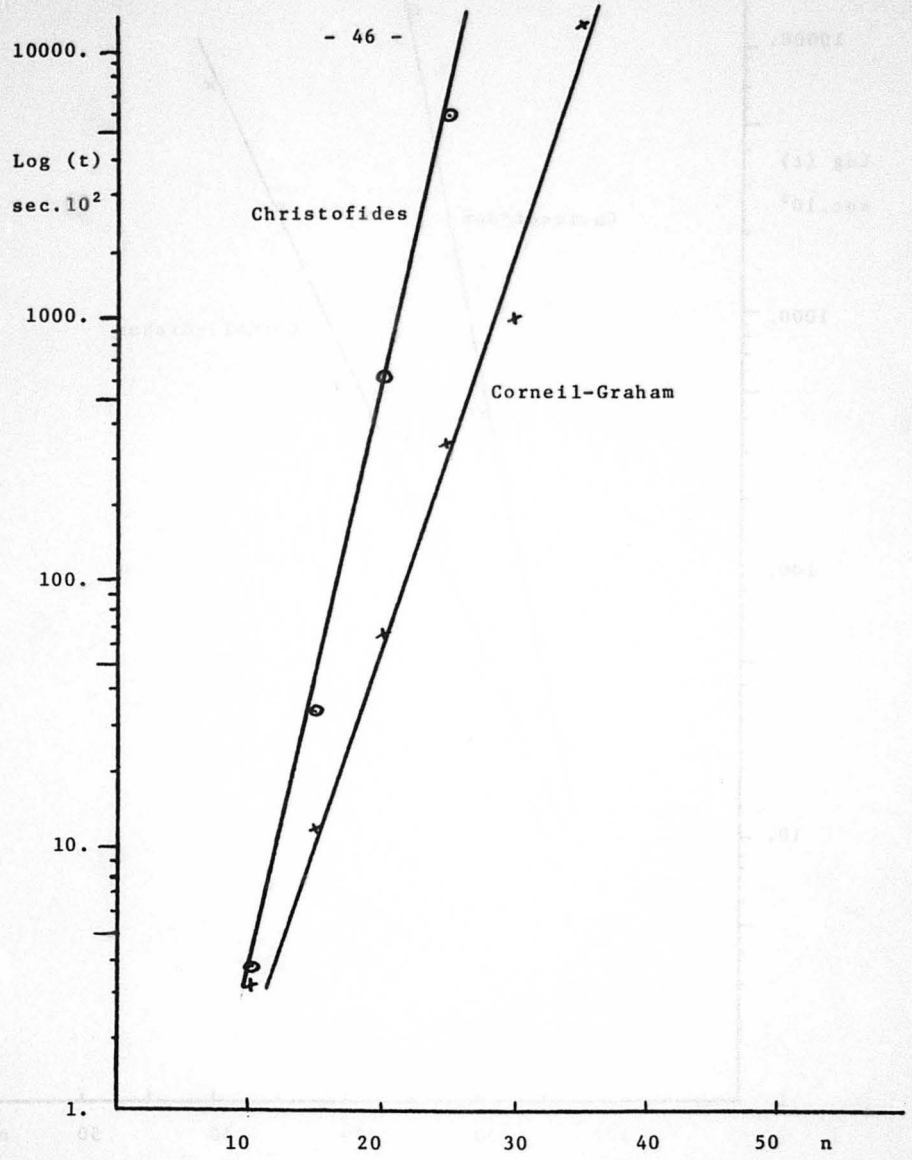


Figure IV.2: Log (time) vs. n. Comparative timings on pseudo-random graphs (ρ=.4).

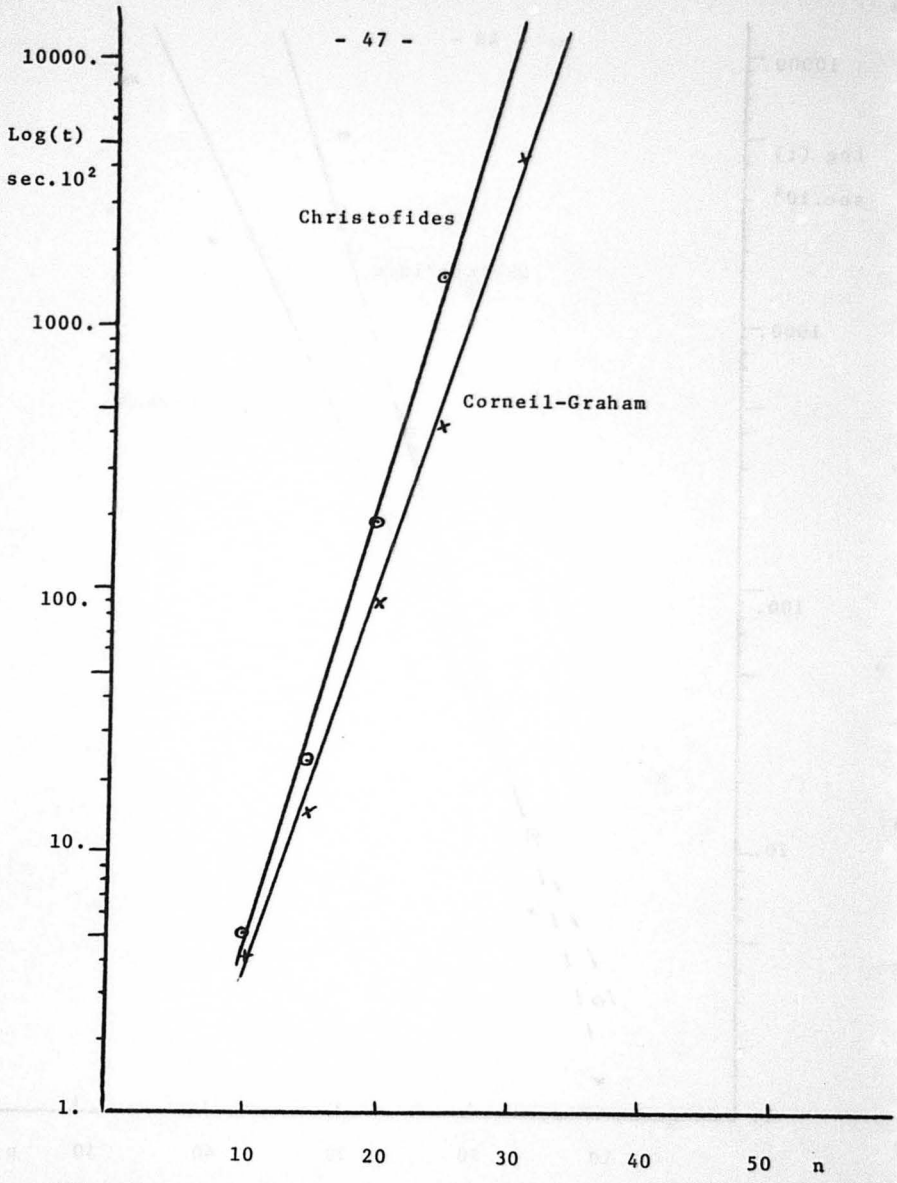


Figure IV.3: Log (time) vs. n. Comparative timings on pseudo-random graphs (ρ=.6)

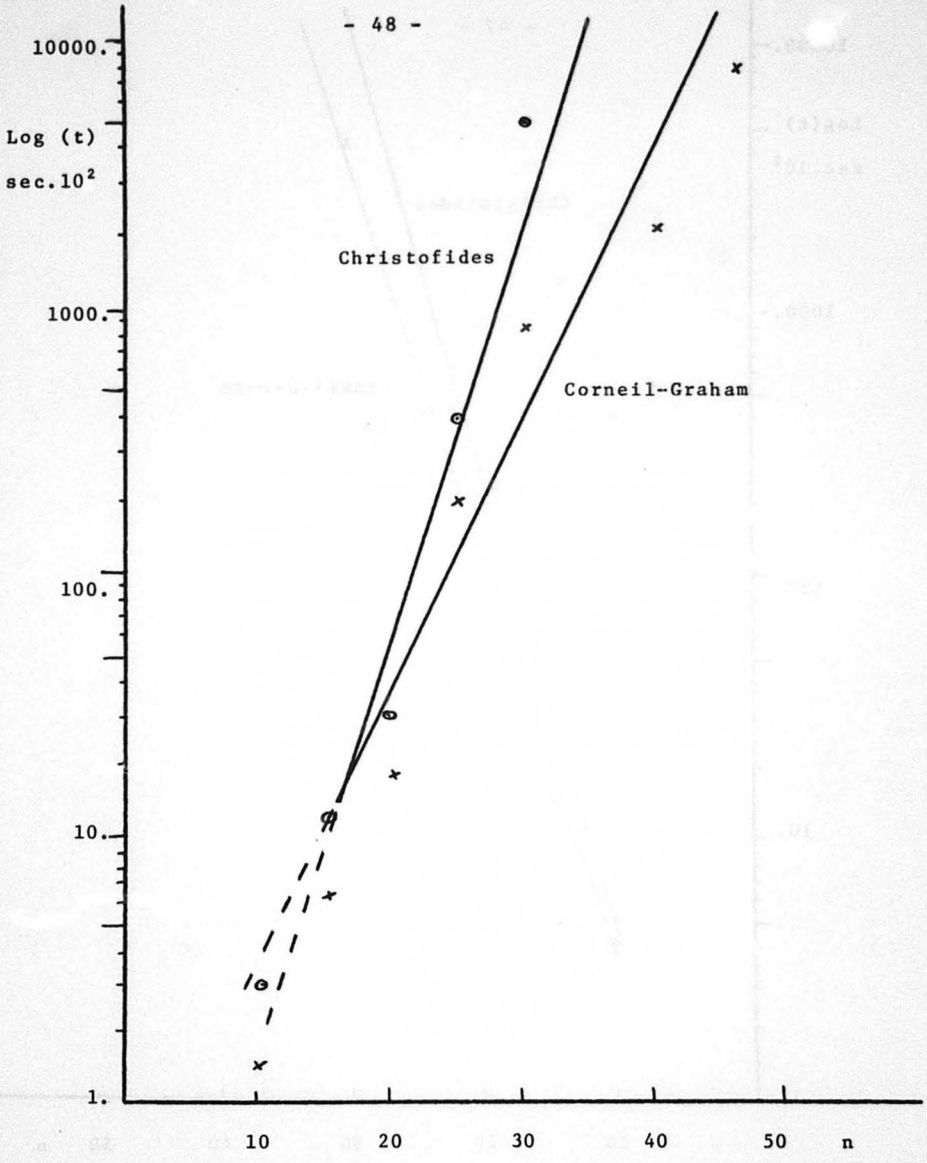


Figure IV.4: Log (time) vs. n. Comparative timings on pseudo-random graphs (ρ=.8).

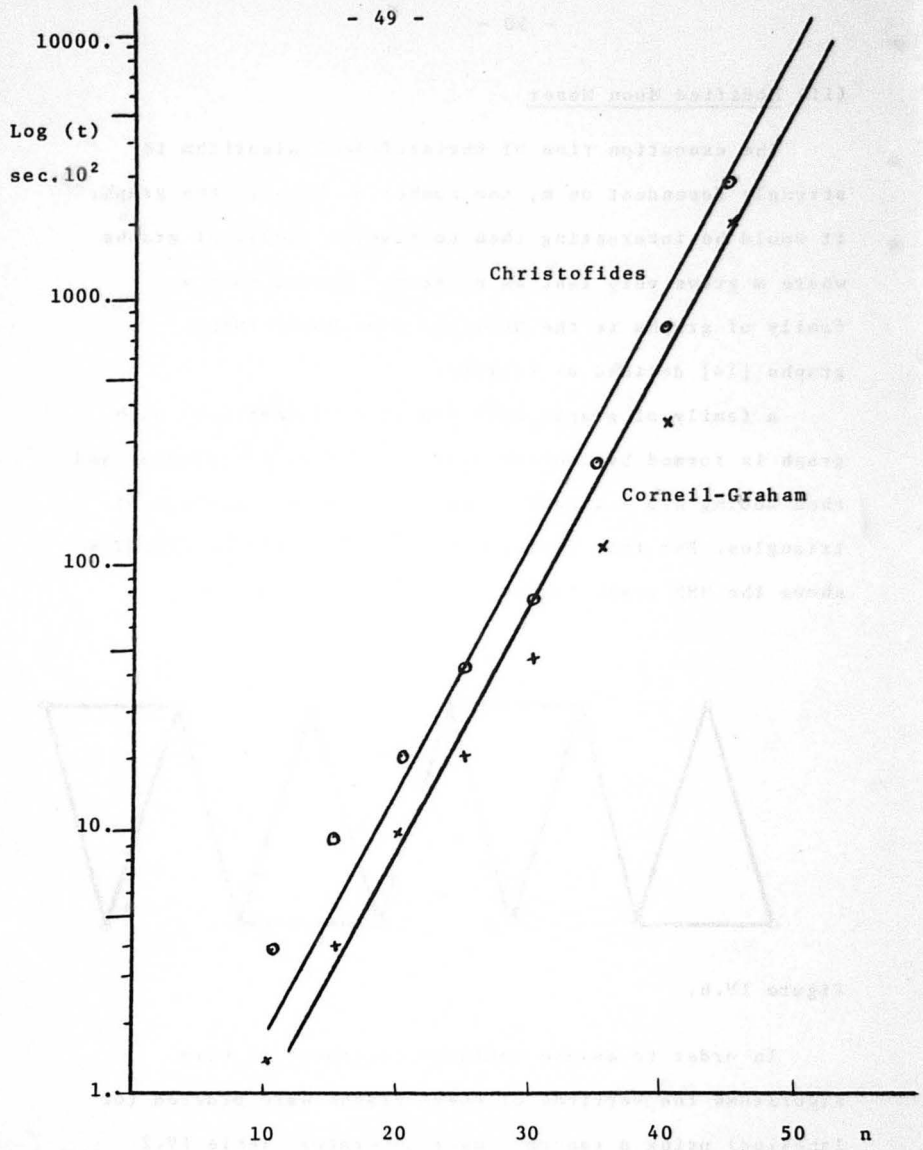


Figure IV.5: Log (time) vs. n. Comparative timings on pseudo-random graphs (ρ=.9).

(ii) Modified Moon Moser

The execution time of Christofides' algorithm is strongly dependent on m, the number of MISS in the graph. It would be interesting then to study a family of graphs where m grows very fast as n becomes large; such a family of graphs is the Modified Moon Moser (MMM) graphs [14] defined as follows:

a family of graphs with n=0 (Mod 3) vertices; each graph is formed by constructing n/3 disjoint triangles and then adding n/3 - 1 additional edges to form a chain of triangles. For this family $m > 2^{n/3}$. For example Fig.IV.6 shows the MMM graph for n=12.

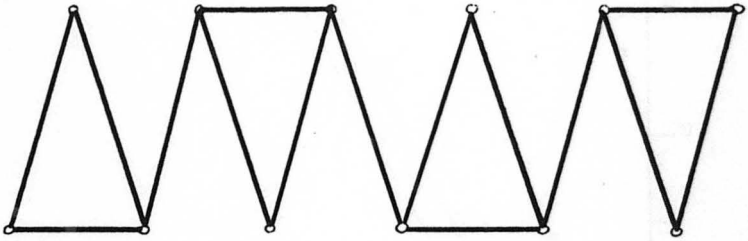


Figure IV.6.

In order to ensure unbiased treatment of both algorithms the vertices of these graphs were ordered (or labelled) using a random number generator. Table IV.2 shows the time and storage requirements for both algorithms while Figure IV.7 displays these results

graphically. Both Table IV.2 and Figure IV.7 demonstrate that for this family of graphs the C-G algorithm is undoubtedly the superior algorithm.

n	time		storage	
	C-G	Christo-fides	C-G	Christo-fides
3	0.0	0.0	8	12
6	1.7	1.7	14	25
9	3.3	8.3	20	60
12	5.0	56.7	26	166
15	8.3	705.0	32	533
18	13.3	8781.7	38	2014
21	18.3	-	44	-
-	-	-	-	-
60	278.3	-	122	-
63	320.0	-	128	-

Table IV.2 Observed time and storage requirements on modified Moon Moser (MMM) graphs.

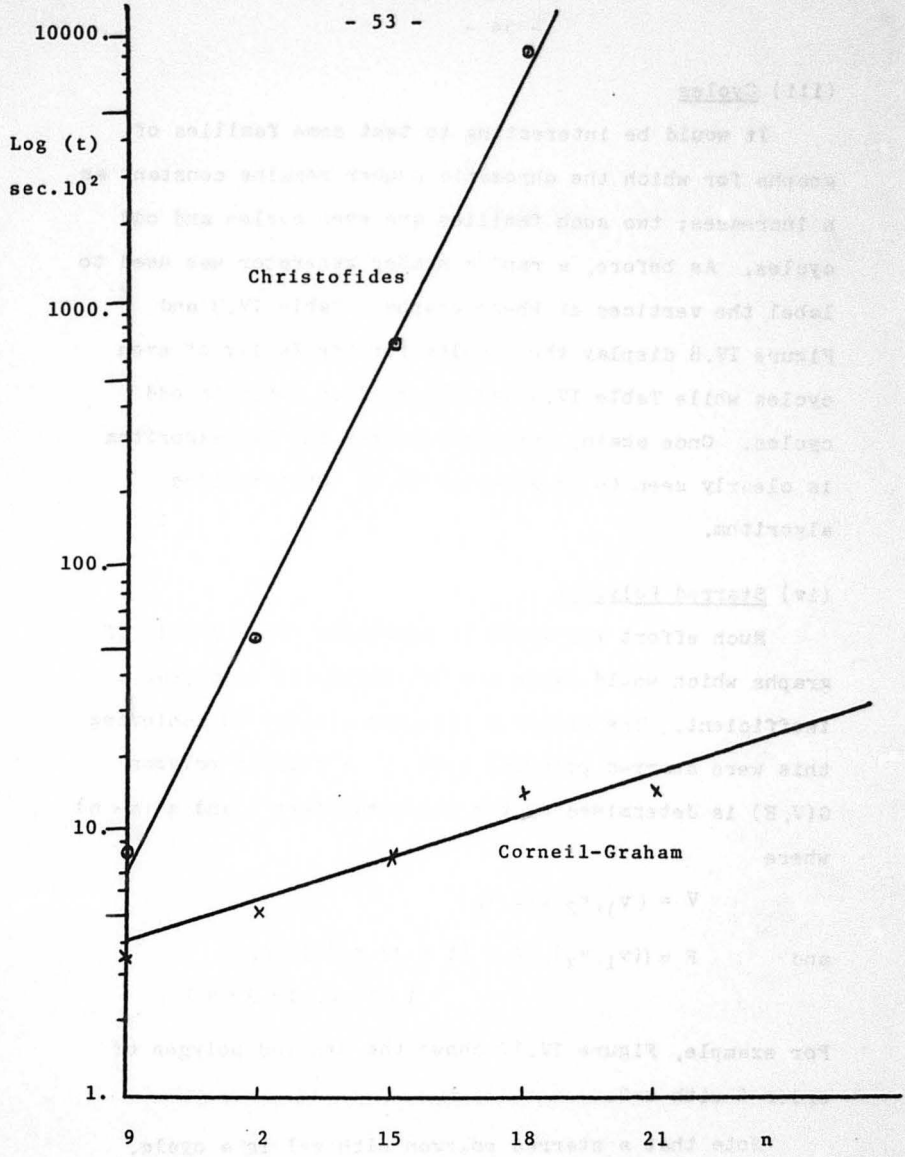


Figure IV.7: Log (time) vs. n. Comparative timings on Modified Moon-Moser graphs.

(iii) Cycles

It would be interesting to test some families of graphs for which the chromatic number remains constant as n increases; two such families are even cycles and odd cycles. As before, a random number generator was used to label the vertices of these graphs. Table IV.3 and Figure IV.8 display the results for the family of even cycles while Table IV.4 and Figure IV.9 refer to odd cycles. Once again, for these graphs the C-G algorithm is clearly seen to be superior to the Christofides algorithm.

(iv) Starred Polygons

Much effort was spent in searching for a family of graphs which would cause the C-G algorithm to appear inefficient. The graphs which came closest to achieving this were starred polygons [19] . A starred polygon G(V,E) is determined by the two parameters n and s (s < n) where

$$V = \{v_1, v_2 \ldots v_n\}$$

and $$E = \{(v_i, v_k),\ k = (i + j) \text{ modulo } n,\ 1 \le i \le n,\ 1 \le j \le s\}\ .$$

For example, Figure IV.10 shows the starred polygon of order 6 with s=2.

Note that a starred polygon with s=1 is a cycle.

n	time		storage	
	C-G	Christo-fides	C-G	Christofides
6	1.7	1.7	14	19
12	5.0	20.0	26	97
18	11.7	1235.0	38	778
24	23.3	-	50	-
-	-	-	-	-
54	185.0	-	110	-
60	243.0	-	122	-

Table IV.3 Observed time and storage requirements on even-length cycles.

n	time		storage	
	C-G	Christo-fides	C-G	Christo-fides
3	0.0	0.0	8	12
9	5.0	5.0	80	42
15	16.7	138.3	224	281
21	48.3	11593.3	440	2862
27	101.7	-	728	-
-	-	-	-	-
63	1320	-	3968	-

Table IV.4 Observed time and storage requirements on odd-length cycles.

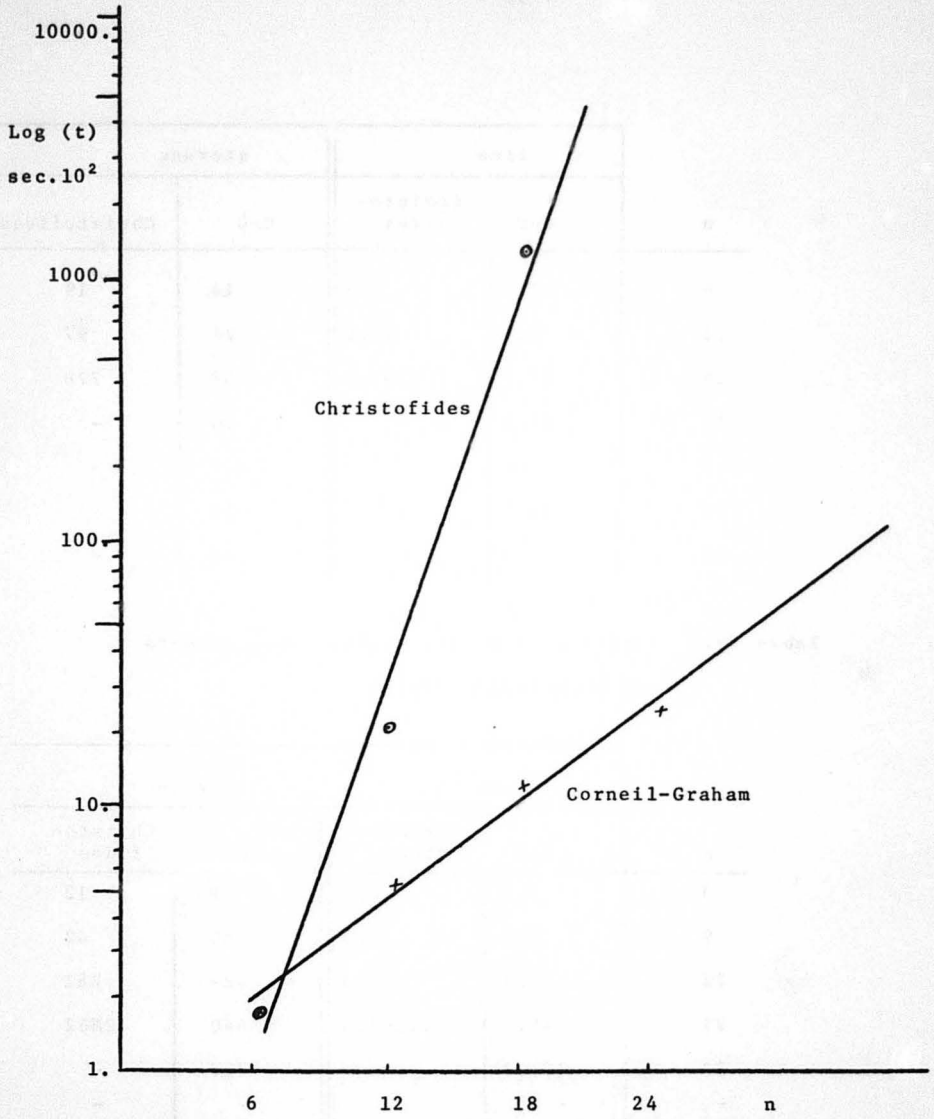


Figure IV.8: Log (time) vs. n. Comparative timings on even cycles.

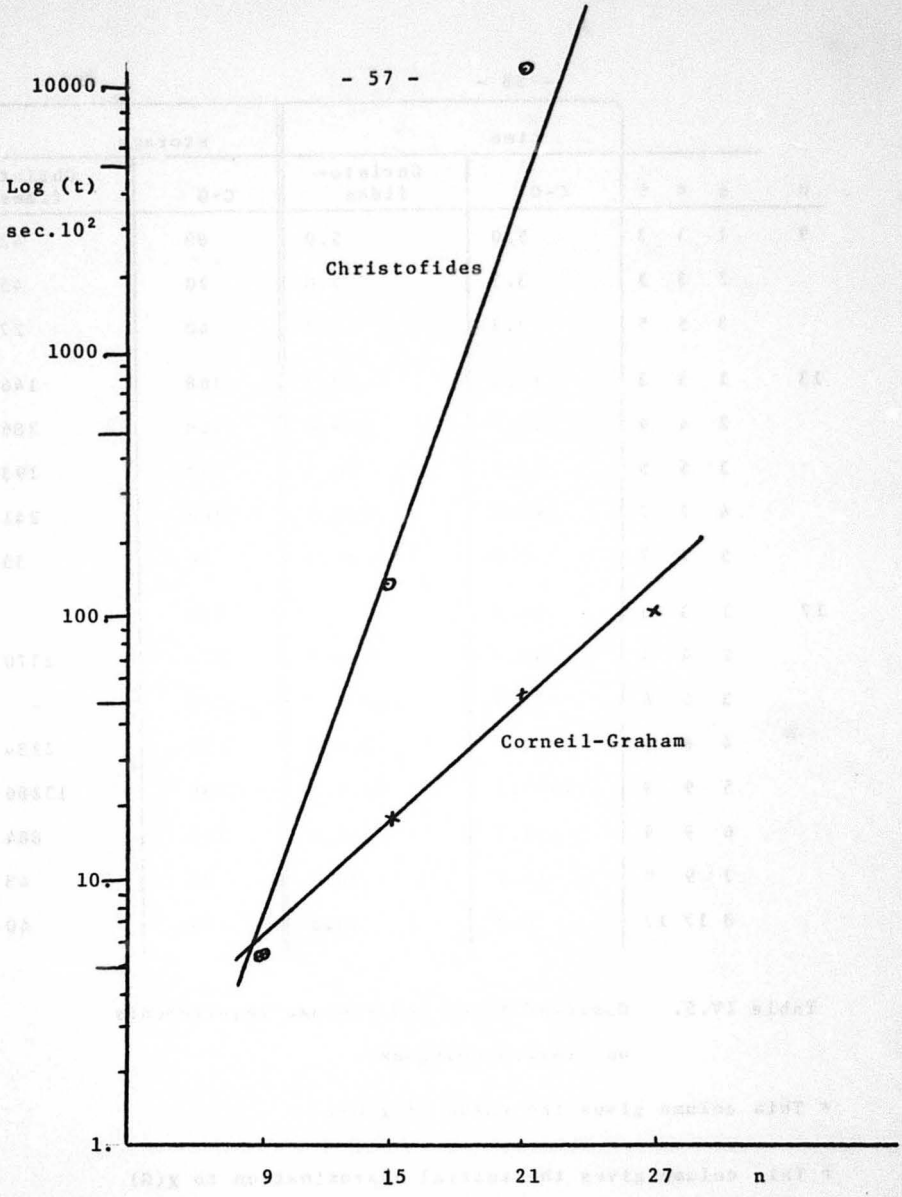


Figure IV.9: Log (time) vs. n. Comparative timings on odd cycles.

				time		storage	
n	s	*	†	C-G	Christo-fides	C-G	Christo-fides
9	1	3	3	5.0	5.0	80	42
	2	3	3	3.3	5.0	20	45
	3	5	5	3.3	3.3	40	27
13	1	3	3	13.3	38.3	168	146
	2	4	4	13.3	100.0	168	286
	3	5	5	13.3	66.7	140	193
	4	7	7	90.0	120.0	168	241
	5	7	7	6.7	10.0	56	35
17	1	3	3	26.7		288	
	2	4	4	31.7	2126.7	324	2170
	3	5	6	73.3	-	360	-
	4	6	6	31.7	1186.7	252	2234
	5	9	9	7043.3	7921.7	288	13286
	6	9	9	551.7	553.0	288	884
	7	9	9	16.7	18.3	72	43
	8	17	17	0.0	18.3	36	40

Table IV.5. Observed times and storage requirements on starred polygons.

* This column gives the value of $\chi(G)$.

† This column gives the initial approximation to $\chi(G)$ as found by the heuristic of section II.4.(i).

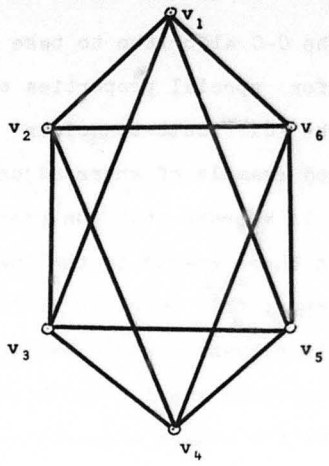


Figure IV.10.

Table IV.5 shows the test results for a series of n and s values. Note that for low values of $s(s < n/3)$ the C-G algorithm is significantly faster than Christofides'. For the graphs (n=13, s=4) and (n=17, s=15) both algorithms are seen to require a substantially increased time for execution. This indicates that some starred polygons are 'difficult to colour' and a study of these graphs should form an important part of any investigation of the colouring problem. For these 'difficult' graphs the C-G algorithm is only slightly faster than Christofides'. For larger values of s $(s > n/3)$ the algorithms have approximately equal execution times.

In section II.5 it was suggested that the user could

make adjustments to the C-G algorithm to take advantage of, or to compensate for, special properties of some classes of graphs. The 'difficult-to-colour' starred polygons provide a good example of where adjustments could be beneficial. It appears that the property which causes difficulty with these graphs is the low ratio of edges to chromatic number; this makes it difficult to build α-cliques. This situation would be improved by always selecting the vertices x and y on which reduction is performed, to have few common adjacencies; this would ensure that few edges would be destroyed by coalescence and would allow a speedier build up of α-cliques with a resultant reduction of the execution time.

IV.3 Conclusions

The data and analysis of the above tests allow us to make the following conclusions and comments.

(i) For all graphs tested the execution time of the C-G algorithm was either significantly less than that of Christofides' or at worst, approximately equal to it.

(ii) For low values of n the storage requirements of the algorithms are approximately equal but for larger graphs Christofides' algorithm requires more storage. This reflects the result in section IV.1 that the storage requirement of the C-G algorithm is $O(n^2)$ while that of

Christofides' algorithm can grow exponentially.

(iii) The asymptotic behaviour (i.e. the slope of Figures IV.1 - IV.5, IV.7 - IV.8) of the C-G algorithm was in most cases the superior of the two algorithms tested. The exception was for pseudo-random graphs with a high density (ρ =.9) of edges; in this case the plots had approximately equal slopes.

In addition, the following observations can be noted:

(iv) For pseudo-random graphs the efficiency of Christofides' algorithm is seen to increase as the density of edges increases from low to high values.

(v) For pseudo-random graphs the C-G algorithm is most efficient when the density of edges is low ($\simeq$ .3) or high ($\simeq$ .9) and is least efficient for intermediate densities ($\simeq$ .6).

(vi) The superiority which the C-G algorithm displays in pseudo-random graphs and starred polygons is most noticeable when a graph has few edges and is least apparent when the graph is almost complete.

REFERENCES

1. Auguston, J.G. and Minker, J. *An Analysis of some Graph Theoretical Cluster Techniques*. J.ACM. 17,4 (Oct. 1970).

2. Berge, C. The Theory of Graphs. London:Methuen (1962).

3. Birkhoff, G.D. *A Determinant Formula for the Number of Ways of Colouring a Map*. Annals of Math.(2) Vol. 14 (1912) pp.42-46.

4. Christofides, N. *An Algorithm for the Chromatic Number of a Graph*. The Computer Journal, Vol. 14 (1971) pp.38-39.

5. Descartes, B. *Solution to Advanced Problem #4526*. Amer. Math. Monthly 61 (1954) p.352.

6. Furtado, Ruschke, dos Santos, Bauer. *Finding the Optimal Colourings of a Graph*. Unpublished report, Departmento de Informatica, Pontificia Universidade Catolica do Rio de Janeiro.

7. Gotlieb, C.C. and Kumar, S. *Semantic Clustering of Index Terms*. J.ACM. 15,4 (Oct. 1968) pp.493-513.

8. Harary, F. Graph Theory. Addison-Wesley (1969).

9. Hedetniemi, S.T. Review #22063, Computing Reviews,12 (Oct. 1971).

10. Karp, R.M. *Reducibility Among Combinatorial Problems*. Technical Report 3 (April 1972) Dept. of Computer Science, University of California.

11. Knuth, D.E. The Art of Computer Programming, Vol. 1, p.316. Addison-Wesley (1968).

12. Lawlor, F.L. and Wood, D.E. *Branch and Bound Methods: a survey*. Operations Research, Vol. 14 (1966) pp.699-719.

13. Minker, J., Wilson, G.A., Zimmerman, B.H. *An Evaluation of Query Expansion by the Addition of Clustered Terms for a Document Retrieval System.* Technical Report TR-172 (Oct. 1971) Computer Science Centre, University of Maryland.

14. Moon, J.W. and Moser, L. *On Cliques in Graphs.* Israel Journal Math.,3 (1965), pp.23-28.

15. Mulligan, G. *Algorithms for Finding Cliques of a Graph.* Technical Report #41, Dept. of Computer Science, University of Toronto.

16. Read, R.C. *An Introduction to Chromatic Polynomials.* J. Combinatorial Theory, 3 (1967) pp. 136-145.

17. Riordan, J. Introduction to Combinatorial Analysis. (1958) Wiley.

18. Slagle, J.R. Artificial Intelligence: The Heuristic Programming Approach. McGraw-Hill (1971).

19. Turner, J. *Point Symmetric Graphs with a Prime Number of Points.* J. Combinatorial Theory, 3 (1967) pp. 136-145.

20. Wagner, R.A. *An Algorithm for Coloring the Nodes of a Graph.* Technical Report , Cornell University (1971).

21. Wells, M.B. Elements of Combinatorial Computing. Pergammon Press (1971).

22. Welsh, D.J.A. and Powell, M.B. *An Upper Bound to the Chromatic Number of a Graph and its Application to Time-Tabling Problems.* The Computer Journal, Vol. 10 (1967), p. 85.

23. Wood, D.C. *A Technique for colouring a Graph Applicable to Large Scale Time-Tabling Problems.* The Computer Journal, Vol. 12 (1969) p.317.

24. Zykov, A.A. *On Some Properties of Linear Complexes.* (Russian) Mat. Sbornik 24 (1949), pp. 163-188. Amer. Math. Soc. Translation N.79 (1952).

APPENDIX I

Corneil - Graham Algorithm

```
**************************************************
*                                                *
*              CORNEIL - GRAHAM                  *
*                                                *
**************************************************
```

```
PROCEDURE CHROMATIC(LOGICAL ARRAY A(*,*)¦INTEGER VALUE N¦INTEGER RESULT
                    ALFA)¦
BEGIN

COMMENT   GIVEN THE ADJACENCY MATRIX A OF A GRAPH OF ORDER N
          THIS PROCEDURE RETURNS ALFA, THE CHROMATIC NUMBER¦

INTEGER PROCEDURE SUMS1(BITS VALUE VCTR)¦

COMMENT  SUMS1 + SUMS2 ADDS THE '1' BITS OF A BITS VECTOR OF LENGTH N¦

BEGIN INTEGER X¦ X¦=0¦
      FOR I¦=0 UNTIL 5N-1 DO
      IF VCTR AND MASK SHL I ¬= ZEROS THEN X¦=X+1¦
      X
END SUMS1¦
INTEGER PROCEDURE SUMS2(BITS VALUE VCTR)¦
BEGIN INTEGER X¦ X¦=0¦
      FOR I¦=0 UNTIL NN-33 DO
      IF VCTR AND MASK SHL I ¬= ZEROS THEN X¦=X+1¦
      X
END SUMS2¦


PROCEDURE DELETE(INTEGER VALUE R¦BITS ARRAY PRESVERTEX(*))¦

COMMENT      TO REMOVE VERTEX R FROM THE SET PRESVERTEX OF VERTICES¦

BEGIN INTEGER T1¦ T1¦=(R-1) DIV 32 +1¦
      PRESVERTEX(T1)¦=PRESVERTEX(T1) AND ¬MASK SHL ((R-1) REM 32)
END DELETE¦
```

```
PROCEDURE REDUCE(BITS ARRAY A(*,*);BITS ARRAY PRESVERTEX(*);
                                    INTEGER VALUE N,NEGA,LEFTS);

COMMENT  SEE SECTION 2.3 OF THESIS
         THE GRAPH A HAS N VERTICES AND NEGA EDGES;

BEGIN
LOGICAL PROCEDURE ADJ(INTEGER VALUE K,L);

COMMENT  RETURNS TRUE IFF I IS ADJACENT TO J IN A;

IF A(K,(L-1)DIV 32+1) AND MASK SHL((L-1)REM 32) =ZEROS THEN FALSE
                                                    ELSE TRUE;


LOGICAL PROCEDURE PRESENT(INTEGER VALUE X);

COMMENT  RETURNS TRUE IFF VERTEX X IS INCLUDED IN GRAPH A;

IF PRESVERTEX((X-1)DIV 32+1) AND MASK SHL((X-1)REM 32) = ZEROS
                                          THEN FALSE ELSE TRUE;
```

```
PROCEDURE CLUSTER;

COMMENT  TO FIND AN ALFA CLUSTER (SEE SECTION 2.4 OF THESIS);

BEGIN INTEGER ARRAY V1(1::N);
INTEGER BETA;  REAL M; LOGICAL TIMES;
TIMES:= FALSE; BETA:= N;  M:= ALFA /2; R:=0;
BEGIN REAL MAXW;  REAL ARRAY W(1::BETA,1::BETA);

                    COMMENT  w IS THE WEIGHT MATRIX
                             MAXW IS THE THRESHOLD VALUE;

              INTEGER ARRAY SAV2(1::BETA);INTEGER V1K,V1L;
              BITS T1,T2;
    FOR I:=1 UNTIL N DO IF PRESENT(I) THEN BEGIN R:=R+1;
                                                 V1(R) := I
                                           END;
R:  MAXW:= 0;
    IF TIMES THEN BEGIN CLUST1:=CLUST2:=ZEROS;
                        FOR I:=1 UNTIL BETA DO
                        IF V1(I)>32 THEN CLUST2:=CLUST2 OR MASK SHL
                                                 (V1(I)-33)
                                    ELSE CLUST1:=CLUST1 OR MASK SHL
                                                 (V1(I)-1)
                  END ELSE
                  BEGIN CLUST1:=PRESVERTEX(1);CLUST2:=PRESVERTEX(2)
                  END;
    FOR K:=2 UNTIL BETA DO
    FOR L:=1 UNTIL K-1 DO
    BEGIN V1K := V1(K); V1L := V1(L);
          T1:=A(V1K,1) AND A(V1L,1) AND CLUST1;
          T2:=A(V1K,2) AND A(V1L,2) AND CLUST2;
          W(K,L):=SUMS1(T1) + SUMS2(T2);
          IF W(K,L)>M THEN W(K,L) := M;
          IF ADJ(V1K,V1L) THEN  W(K,L) := W(K,L) + M + .5;
          IF W(K,L) > MAXW THEN MAXW := W(K,L);
    END;
              COMMENT  TO ENSURE THAT THERE ARE AT LEAST ALFA
                       EDGES OF WEIGHT >= THE THRESHOLD MAXw;
    BEGIN INTEGER CARD;
C:  CARD:= 0;
    FOR K:=2 UNTIL BETA DO
    FOR L:=1 UNTIL K-1 DO
    IF W(K,L) >=MAXW THEN CARD := CARD + 1;
    IF CARD < ALFA THEN BEGIN MAXW:= MAXW*.9;
                              GO TO C
                        END;
    END;
```

```
BEGIN LOGICAL ARRAY T(1::BETA,1::BETA); LOGICAL ARRAY INCLUD(1::BETA);
      INTEGER P,S,S1,J,I,TINCLUD,J1;  INTEGER ARRAY STACK(0::1,1::BETA);
                                      INTEGER ARRAY SAV1(1::BETA);

        COMMENT  T IS THE THRESHOLD MATRIX;

      S1 := BETA + 1; MAXW:= MAXW -1*-5;
   C: FOR K:=1 UNTIL BETA DO
      BEGIN INCLUD(K):= FALSE; W(K,K):=0;
            FOR L:=1 UNTIL K-1 DO
            T(K,L):=T(L,K):= IF W(K,L) >=MAXW THEN TRUE ELSE FALSE
      END;

      COMMENT  TO FIND THE CONNECTED COMPONENTS OF T;

      TINCLUD := 0;  J1 := 1;J:=0;
  D1: STACK(J,1):=J1; INCLUD(J1):= TRUE; S:=1; P:=0;
   D: P:=P+1; I:=STACK(J,P);
      FOR K:= 2 UNTIL BETA DO
      IF ¬INCLUD(K) AND T(I,K) THEN BEGIN S:=S+1; STACK(J,S):=K;
                                          INCLUD(K):= TRUE
                                    END;
      IF S > P THEN GO TO D;
      TINCLUD:=TINCLUD + S;
      IF (S>=ALFA) AND(S<=S1) THEN BEGIN S1:=S; J:=(J+1) REM 2
                                   END;
      IF TINCLUD <=BETA - ALFA THEN FOR K:=1 UNTIL BETA DO
      IF ¬INCLUD(K) THEN BEGIN J1:=K; GO TO D1;
                         END;
      IF S1=BETA+1 THEN BEGIN MAXW:=(MAXW*9)/10;
                              GO TO C
                        END;
      BETA:=S1; J:=(J+1) REM 2;
      FOR K:=1 UNTIL BETA DO  BEGIN SAV2(K) := STACK(J,K);
                                    SAV1(K):=V1(SAV2(K))
                              END;
      FOR K:=1 UNTIL BETA DO V1(K):= SAV1(K);
      IF (BETA<N) AND (BETA>ALFA*(3/2)) AND ¬TIMES THEN
                                    BEGIN TIMES:=TRUE;
                                          GO TO B
                                    END
END;
```

```
IF BETA > ALFA THEN
BEGIN REAL ARRAY STRENGTH(1::BETA);LOGICAL ARRAY INCLUD(1::BETA);
      INTEGER P,MNODE,MINODE,SAV2P; REAL MAXS,MINS;
      FOR K:=1 UNTIL BETA DO BEGIN STRENGTH(K):=0; INCLUD(K):=FALSE
                               END;
      FOR K:=2 UNTIL BETA DO
      FOR L:=1 UNTIL K-1 DO BEGIN
         IF SAV2(K)<SAV2(L) THEN W(SAV2(K),SAV2(L)):=W(SAV2(L),SAV2(K));
         STRENGTH(K):=STRENGTH(K) + W(SAV2(K),SAV2(L));
         STRENGTH(L):=STRENGTH(L) + W(SAV2(K),SAV2(L));
                              END;
      MAXS:=-1; MINS:=MAXW*10;
      IF BETA ¬= ALFA+1 THEN
BEGIN FOR K:=1 UNTIL BETA DO
      IF STRENGTH(K)>MAXS THEN  BEGIN MAXS:=STRENGTH(K);
                                      MNODE:=K
                                END
END                        ELSE
BEGIN       FOR K:=1 UNTIL BETA DO
            IF STRENGTH(K)<MINS THEN
                                BEGIN MINS:=STRENGTH(K); MINODE:=K
                                END;
            FOR K:=1 UNTIL ALFA DO
            V(K):=IF K=MINODE THEN V1(BETA) ELSE V1(K);
            GO TO XIT
END:
      V(1):=V1(MNODE); INCLUD(MNODE):=TRUE;  P:=MNODE; SAV2P:=SAV2(P);
      FOR K:=1 UNTIL BETA DO STRENGTH(K):=0;
      FOR J:=2 UNTIL ALFA DO
      BEGIN MAXS:=-1;
            FOR I:=1 UNTIL BETA DO
            IF ¬INCLUD(I) THEN BEGIN STRENGTH(I):=STRENGTH(I) +
           (IF I>P THEN W(SAV2(I),SAV2P  ) ELSE W(SAV2P  ,SAV2(I)));
                                     IF STRENGTH(I)>MAXS THEN
                                     BEGIN MAXS:=STRENGTH(I); MNODE:=I
                                     END
                               END;
            V(J):=V1(MNODE);INCLUD(MNODE):=TRUE ;P:=MNODE;SAV2P:=SAV2(P)
      END
END
ELSE FOR K:=1 UNTIL ALFA DO V(K):=V1(K);
XIT: END
END CLUSTER;
```

```
INTEGER ARRAY DEG, V(1::ALFA); INTEGER SALFA,NEGAV,NS1,R;

COMMENT ARRAY V WILL CONTAIN THE ALFA CLUSTER
        DEG(I) IS THE DEGREE OF VERTEX I IN THE ALFA CLUSTER
        NEGAV IS THE NUMBER OF EDGES IN THE ALFA CLUSTER;

BITS CLUST1,CLUST2;
BEGIN LOGICAL FINA; INTEGER MEGA;

COMMENT  TO CHECK IF REDUCTION OF A IS REQUIRED
         IF REDUCTION IS NECESSARY THEN FINA WILL BE 'TRUE';

B:     FINA:=TRUE; MEGA := N*(N-1);
       IF ALFA>N THEN BEGIN ALFA:=N; MEGFA :=MEGA
                      END;
       IF NEGA = MEGA THEN FINA:=FALSE
                      ELSE
       IF NEGA<= MEGA - 2 THEN BEGIN IF ALFA > N-1 THEN
                                     BEGIN ALFA:=N-1;MEGFA:=(N-1)*(N-2)
                                     END;
                                     IF NEGA=MEGA-2 THEN FINA:=FALSE
                               END;

                          COMMENT THIS IS HEURISTIC 3 OF SECTION 2.4;
       IF (LEFTS<3) AND FINA THEN
       FOR K:=2 UNTIL NN DO IF PRESENT(K) THEN
       FOR L:=1 UNTIL K-1 DO IF PRESENT(L) THEN
       BEGIN IF(A(K,1) AND ¬A(L,1) AND PRESVERTEX(1)) OR
                (A(K,2) AND ¬A(L,2) AND PRESVERTEX(2)) = ZEROS THEN
             BEGIN NEGA:=NEGA -2*(SUMS1(A(K,1) AND PRESVERTEX(1)) +
                                  SUMS2(A(K,2) AND PRESVERTEX(2)));
                   DELETE(K,PRESVERTEX); N:=N-1;
             END                                                ELSE
             IF(A(L,1) AND ¬A(K,1) AND PRESVERTEX(1)) OR
               (A(L,2) AND ¬A(K,2) AND PRESVERTEX(2)) = ZEROS THEN
             BEGIN NEGA:=NEGA -2*(SUMS1(A(L,1) AND PRESVERTEX(1)) +
                                  SUMS2(A(L,2) AND PRESVERTEX(2)));
                   DELETE(L,PRESVERTEX); N:=N-1;
             END   ELSE GO TO TRYNEXT;
             GO TO B;
TRYNEXT:
       END;
       IF ¬FINA THEN GO TO XITPAND
END CHECK;
       SALFA := ALFA;  COMMENT TO STORE THE PRESENT VALUE OF ALFA
                               AS A LOCAL VARIABLE SALFA;
       CLUSTER;
       CLUST1:=CLUST2:=ZEROS;
       FOR I:=1 UNTIL ALFA DO
       IF V(I)>32 THEN CLUST2:=CLUST2 OR MASK SHL (V(I)-33)
                  ELSE CLUST1:=CLUST1 OR MASK SHL (V(I)-1);
       WHILE (R<=ALFA) AND (V(R)<33) DO
       BEGIN CLUST1:=CLUST1 OR MASK SHL (V(R)-1); R:=R+1
       END;
        FOR I:=R UNTIL ALFA DO CLUST2:=CLUST2 OR MASK SHL (V(I)-33);
       NEGAV:=0;
       FOR K:=1 UNTIL ALFA DO BEGIN DEG(K):=SUMS1(A(V(K),1) AND CLUST1) +
                                            SUMS2(A(V(K),2) AND CLUST2);
                                    NEGAV:=NEGAV + DEG(K)
                              END;
```

```
LOOP: IF NEGAV < MEGFA THEN
      BEGIN PROCEDURE FORMB;

      COMMENT  TO FORM A GRAPH  B  BY COALESCENCE
               B WILL HAVE NEGB EDGES;

            BEGIN NEGB:=NEGA;
            FOR K:=1 UNTIL NN DO BEGIN B(K,1):=A(K,1);B(K,2):=A(K,2)
                                 END;
            B(X,1):=B(X,1) OR B(Y,1);  B(X,2):=B(X,2) OR B(Y,2);
            PRESVERTB(1):=PRESVERTEX(1); PRESVERTB(2):=PRESVERTEX(2);
            NEGB:=NEGB-2*(SUMS1(A(X,1) AND A(Y,1) AND PRESVERTEX(1)) +
                          SUMS2(A(X,2) AND A(Y,2) AND PRESVERTEX(2)));
            DELETE(Y,PRESVERTB);
            BEGIN INTEGER I1,T2; T1:=(X-1)DIV 32+1; T2:=(X-1)REM 32;
                  FOR I:=0 UNTIL SN-1 DO
                  IF A(Y,1) AND MASK SHL I ¬= ZEROS THEN
                  B(I+1,T1):=B(I+1,T1) OR MASK SHL T2;
                  FOR I:=0 UNTIL NN-33 DO
                  IF A(Y,2) AND MASK SHL I ¬= ZEROS THEN
                  B(I+33,T1):=B(I+33,T1) OR MASK SHL T2
            END
            END FORMB;
```

```
BITS ARRAY B(1::NN,1::2);BITS ARRAY PRESVERTB(1::2);
INTEGER NEGB,MIN1,MIN2,X,Y,T1,T2;
IF LEFTS> 2 THEN BEGIN MIN1:=MIN2:=N;

                COMMENT 'LEFTS>2' IS A STRATEGY FOR CHOOSING
                        THE VERTICES X AND Y ON WHICH
                        COALESCENCE IS PERFORMED;

                         FOR K:=1 UNTIL ALFA DO
                         IF DEG(K)<MIN1 THEN
                         BEGIN  MIN1:=DEG(K); X:=K
                         END;
                         DEG(X):=DEG(X)+1;
                         X:=V(X);
                         FOR K:=1 UNTIL ALFA DO
    IF (DEG(K)<MIN2) AND ¬ADJ(X,V(K)) AND (X¬=V(K)) THEN
                         BEGIN  MIN2:=DEG(K); Y:=K
                         END;
                         DEG(Y):=DEG(Y)+1;
                         Y:=V(Y)
                    END
                ELSE BEGIN
                         MIN1:=MIN2:=-1;
                         FOR K:=1 UNTIL ALFA DO
                    IF (DEG(K)>MIN1) AND (DEG(K)<ALFA-1) THEN
                         BEGIN  MIN1:=DEG(K); X:=K
                         END;
                         DEG(X):=DEG(X)+1;
                         X:=V(X);
                         FOR K:=1 UNTIL ALFA DO
    IF (DEG(K)>MIN2) AND ¬ADJ(X,V(K)) AND (X¬=V(K)) THEN
                         BEGIN  MIN2:=DEG(K); Y:=K
                         END;
                         DEG(Y):=DEG(Y)+1;
                         Y:=V(Y)
                    END;
DEPTH:=DEPTH+1;IF DEPTH>MAXDEPTH THEN MAXDEPTH:=DEPTH;
FORMB;
T1:=(Y-1) DIV 32 +1;  T2:= (X-1) DIV 32 +1;
A(X,T1):=A(X,T1) OR MASK SHL((Y-1) REM 32);
A(Y,T2):=A(Y,T2) OR MASK SHL((X-1) REM 32);
NEGAV:=NEGAV+2; NEGA:=NEGA+2;
NS1:=N-1;
REDUCE(B,PRESVERTB,NS1,NEGB,LEFTS);
IF SALFA= ALFA THEN BEGIN LEFTS:=LEFTS+1; GO TO LOOP
                    END
                    ELSE REDUCE(A,PRESVERTEX,N,NEGA,LEFTS+1)
    END;
XITPAND:
DEPTH:=DEPTH-1;
END REDUCE;
```

```
INTEGER NEGA,MEGFA,NN,SN,DEPTH,MAXDEPTH;
COMMENT  GRAPH A HAS N=NN VERTICES AND NEGA EDGES;

BITS ARRAY   B(1::N,1::2); BITS ARRAY PRESVERTEX(1::2);
BITS ZEROS,MASK;

COMMENT  TO SET THE BITS MATRIX B EQUAL TO
         THE GIVEN ADJACENCY MATRIX A;

NN:=N; SN:= IF N>32 THEN 32 ELSE N;
ZEROS:=#0; MASK:=#1;
FOR K:=1 UNTIL N DO
BEGIN
      B(K,1):=B(K,2):=ZEROS;
      FOR L:=1 UNTIL SN DO IF A(K,L) THEN
      B(K,1):=B(K,1) OR MASK SHL (L-1);
      FOR L:=33 UNTIL N DO IF A(K,L) THEN
      B(K,2):=B(K,2) OR MASK SHL (L-33)
END;
NEGA:=0;
PRESVERTEX(1):=(¬ZEROS) SHR (32-SN);
PRESVERTEX(2):=(¬ZEROS) SHR (64-N);
FOR K:=1 UNTIL N DO
NEGA:=NEGA+SUMS1(B(K,1))+SUMS2(B(K,2));

COMMENT     TO FIND THE INITIAL APPROXIMATION ALFA (SEE SECTION 2.4 OF
                                                          THESIS);

BEGIN
PROCEDURE DIVY (INTEGER VALUE D1,D2);
BEGIN

COMMENT  TO COALESCE VERTICES D1 AND D2;

FOR L:=D1+1 UNTIL N DO    A(D1,L):=A(L,D1);
FOR L:=D2+1 UNTIL N DO    A(D2,L):=A(L,D2);
FOR K:=2 UNTIL N DO
IF A(D1,K) OR A(D2,K) THEN
BEGIN FOR L:=1 UNTIL K-1DO
      IF ¬A(K,L) THEN
      IF A(D1,L) AND A(D2,L)
 THEN BEGIN IF A(D1,K) AND A(D2,K) THEN
            SIMLY(K,L):=SIMLY(K,L)-1
      END
 ELSE IF (A(D1,L) OR A(D2,L) ) AND ¬( A(D1,K) AND A(D2,K) )
                                    AND ¬(A(D1,L)=A(D1,K)) THEN
      SIMLY(K,L) := SIMLY(K,L)+1;
      IF D1>K THEN SIMLY(D1,K):=0 ELSE SIMLY(K,D1) :=0;
END
ELSE FOR L:=1 UNTIL K-1 DO
     IF A(D2,L) AND A(K,L) AND ¬A(D1,L) THEN
     IF D1>K THEN SIMLY(D1,K) := SIMLY(D1,K) +1
            ELSE SIMLY(K,D1) := SIMLY(K,D1) +1;
     FOR L:= 1 UNTIL D1-1 DO
     A(D1,L) := A(D1,L) OR A(D2,L);
     FOR K:=D1+1 UNTIL N DO
     A(K,D1) := A(K,D1) OR A(D2,K);
     DELETE(D2);
     CHROM
END DIVY;
```

```
PROCEDURE CHROM;
BEGIN

COMMENT  TO FIND A MAXIMAL ELEMENT OF THE SIMILARITY MARRIX SIMLY;

INTEGER R,S,OLDSIM;
LEAF;
OLDSIM:= -1;
FOR K:=2 UNTIL N DO
FOR L:=1 UNTIL K-1 DO
IF ¬A(K,L) AND (SIMLY(K,L)>OLDSIM) THEN
BEGIN R:=K; S:=L; OLDSIM:=SIMLY(K,L); IF OLDSIM = N-2 THEN
      BEGIN  FOR K:=2 UNTIL N DO
             FOR L:=1 UNTIL K-1 DO
             IF ¬A(K,L) THEN SIMLY(K,L):= SIMLY(K,L)-1;
             DELETE(R); CHROM
      END
END;
IF OLDSIM=-1 THEN
BEGIN ALFA:=N; GO TO XIT;
END;
DIVY(S,R)
END CHROM;


PROCEDURE LEAF;
BEGIN INTEGER DEGREE;

COMMENT  A HEURISTIC TO REMOVE ANY LEAF-VERTEX BEFORE COALESCENCE;

XITT; IF N<3 THEN
BEGIN ALFA:= CASE N OF (1,IF A(2,1) THEN 2 ELSE 1);  GO TO XIT
END;
FOR K:=1 UNTIL N DO
BEGIN DEGREE:=0;
      FOR L:=1 UNTIL K-1 DO
      IF A(K,L) THEN
      BEGIN DEGREE:=DEGREE+1; IF DEGREE=2 THEN GO TO TRYNEXT
      END;
      FOR L:=K+1 UNTIL N DO
      IF A(L,K) THEN
      BEGIN DEGREE:=DEGREE+1; IF DEGREE=2 THEN GO TO TRYNEXT
      END;
      DELETE(K); GO TO XITT;
TRYNEXT;
END
END LEAF;


PROCEDURE DELETE(INTEGER VALUE R);

COMMENT  TO REMOVE A VERTEX FROM A(E,G, AFTER COALESCENCE);

BEGIN  FOR L:=1 UNTIL R-1 DO
BEGIN  A(R,L):= A(N,L); SIMLY(R,L):=SIMLY(N,L)
END;    FOR K:= R+1 UNTIL N-1 DO
BEGIN   A(K,R):= A(N,K); SIMLY(K,R):=SIMLY(N,K)
END; N:=N-1
END DELETE;
```

```
INTEGER SIM;
INTEGER ARRAY SIMLY(1::N,1::N);

COMMENT  TO SET UP THE SIMILARITY MATRIX SIMLY;

FOR K:=2 UNTIL N DO
FOR L:=1 UNTIL K-1 DO
IF A(K,L) THEN SIMLY(K,L):=0
          ELSE BEGIN SIM:=0;
                     FOR KL:=1 UNTIL N DO
                     IF A(K,KL) AND A(L,KL) THEN SIM:=SIM+1;
                     SIMLY(K,L):=SIM
                END ;
CHROM;
XIT:
MEGFA:=ALFA*(ALFA-1);
WRITE ("INITIAL  APPROXIMATION TO CHROMATIC  NUMBER IS ", ALFA)
END;
N:=NN;
DEPTH:=MAXDEPTH:=1;
REDUCE(B,PRESVERTEX,N,NEGA,0);
WRITE ("EXACT VALUE OF CHROMATIC NUMBER IS              ",ALFA);
END CHROMATIC;
```

APPENDIX II

The Christofides Algorithm

****
* *
* CHRISTOFIDES *
* *
****

```
PROCEDURE CHROMATIC(LOGICAL ARRAY A(*,*);INTEGER VALUE N;INTEGER RESULT
                    ALFA);
BEGIN

COMMENT   GIVEN A, THE COMPLIMENT OF THE ADJACENCY MATRIX OF A GRAPH
          OF ORDER N THIS PROCEDURE RETURNS ALFA, THE CHROMATIC NUMBER;
```

```
PROCEDURE EXTEND(INTEGER ARRAY OLD(*); INTEGER VALUE NE,CE);


COMMENT  THIS IS THE BRON-KERBOSCH CLIQUE-ALGORITHM;


BEGIN INTEGER ARRAY NEW(1::CE);
INTEGER NOD,FIXP,NEWNE,NEWCE,I,J,COUNT,POS,P,S,SEL,MINNOD;
MINNOD:=CE; NOD:=0;
FOR I:=1 UNTIL CE DO IF MINNOD¬=0 THEN
BEGIN P:=OLD(I); COUNT:=0; J:=NE+1;
      WHILE (J<= CE) DO BEGIN
                              IF ¬A(P,OLD(J)) THEN BEGIN COUNT:=COUNT+1;
                                                         POS:=J
                                                   END;
                              J:=J+1
                        END;
      IF COUNT < MINNOD THEN BEGIN FIXP:=P; MINNOD:=COUNT;
                                   IF I<=NE THEN S:=POS
                                            ELSE BEGIN S:=I; NOD:=1
                                                 END
                              END;
END;
NOD:=MINNOD + NOD;  WHILE NOD>0 DO
BEGIN
      P:=OLD(S); OLD(S):=OLD(NE+1); SEL:=OLD(NE+1):=P; NEWNE:=0; I:=1;
      WHILE I<=NE DO BEGIN P:=OLD(I);
                           IF A(SEL,P) THEN BEGIN NEWNE:=NEWNE+1;
                                                  NEW(NEWNE):=P
                                            END;
                        I:=I+1
                     END;
      NEWCE:=NEWNE; I:=I+1;
      WHILE I<=CE DO BEGIN P:=OLD(I);
                           IF A(SEL,P) THEN BEGIN NEWCE:=NEWCE+1;
                                                  NEW(NEWCE):=P
                                            END;
                        I:=I+1
                     END;
       C:=C+1;  COMPSUB(C):=SEL;
       IF NEWCE=0 THEN BEGIN BITS TEMP1,TEMP2,MASK;
                             MASK := #1; TEMP2:=TEMP1:= #0;
                             FOR K:= 1 UNTIL C DO
                             IF COMPSUB(K)>32 THEN
                             TEMP2:=TEMP2 OR MASK SHL (COMPSUB(K)-33)
                                          ELSE
                             TEMP1:=TEMP1 OR MASK SHL (COMPSUB(K)- 1);
                             M:=M+1;
                             ALLMISS := MISLINK(TEMP2,TEMP1,ALLMISS);
                       END
                  ELSE IF NEWNE<NEWCE THEN EXTEND(NEW,NEWNE,NEWCE);
      C:=C-1; NE:=NE+1; NOD:=NOD-1;
      IF NOD>0 THEN BEGIN S:=NE+1; WHILE  A(FIXP,OLD(S)) DO S:=S+1
                    END
END;
FULS := (M-1) DIV 32; XTRA := (M-1) REM 32 + 1;
END EXTEND;
```

```
INTEGER ARRAY ALL,COMPSUB(1::N);
INTEGER C,M,FULS,XTRA;
RECORD MISLINK(BITS MISS2,MISS1; REFERENCE(MISLINK)LINK);
REFERENCE (MISLINK) ALLMISS,TEMP;
M:=0;
ALLMISS:=NULL;
FOR C:=1 UNTIL N DO ALL(C):=C; C:=0;
EXTEND(ALL,0,N);
BEGIN BITS ARRAY MISS(1::M,1::2);  COMMENT  THE M  MISS OF THE GRAPH;
      BITS ARRAY PRESVERTEX(1:: 8000,1::2);   COMMENT  THE UNCOLOURED
                                              VERTICES OF EACH STATE;
      BITS ARRAY STATE(1:: 8000,1::FULS+1);   COMMENT  THE UNCOLOURED
                                              MISS AT EACH STEP;
      LOGICAL ARRAY TEMP1,TEMP2(1::M);
      INTEGER SN,COVER,OLD,NEW,X,J,CHR,LU;    COMMENT  CHR WILL BE
                                              THE CHROMATIC NUMBER;
      BITS ZEROS,MASK,ONES;
LOGICAL PROCEDURE VERTMISS(INTEGER VALUE X,Y);

         COMMENT  TRUE IFF VERTEX X IS IN MISS Y;

IF MISS(Y,(X-1) DIV 32 + 1) AND MASK SHL ((X-1) REM 32)=ZEROS THEN FALSE
                                                        ELSE TRUE;

LOGICAL PROCEDURE STATVERT(INTEGER VALUE S,X);

         COMMENT  TRUE IFF VERTEX X IS PRESENT IN STATE S;

IF PRESVERTEX(S,(X-1) DIV 32 + 1) AND MASK SHL((X-1)REM 32) = ZEROS
                                              THEN FALSE ELSE TRUE;

LOGICAL PROCEDURE STATMISS(INTEGER VALUE J);

         COMMENT  TRUE IFF MISS J IS IN STATE OLD;

IF STATE(OLD,(J-1) DIV 32 + 1) AND MASK SHL ((J-1) REM 32) = ZEROS
                                              THEN FALSE ELSE TRUE;

LOGICAL PROCEDURE SUBSET(INTEGER VALUE Y,Z);

         COMMENT TRUE IFF MISS Y IS A SUBSET OF MISS Z;

IF (PRESVERTEX(OLD,1) AND (¬MISS(J,1)) AND MISS(Y,1) AND(¬MISS(Z,1))) OR
   (PRESVERTEX(OLD,2) AND (¬MISS(J,2)) AND MISS(Y,2) AND(¬MISS(Z,2)))
                          = ZEROS THEN TRUE ELSE FALSE;
LOGICAL PROCEDURE INDEPENDENT(INTEGER VALUE Y,Z);
         COMMENT TRUE IFF MISS Y AND Z HAVE NO VERTEX IN COMMON;
IF (PRESVERTEX(OLD,1) AND MISS(Y,1) AND MISS(Z,1)) OR
   (PRESVERTEX(OLD,2) AND MISS(Y,2) AND MISS(Z,2)) =ZEROS THEN TRUE
                                                   ELSE FALSE;
```

```
INTEGER PROCEDURE MINVERTEX;

        COMMENT  TO FIND THE VERTEX WHICH APPEARS IN FEWEST MISS
                 COVER  IS THE # OF MISS WHICH CONTAIN THIS VERTEX;

BEGIN INTEGER  X,MIN,CNT;
      MIN := M+1;
      FOR I:=1 UNTIL N DO
      IF STATVERT(OLD,I) THEN
      BEGIN CNT:=0;
            FOR J:=1 UNTIL M DO IF STATMISS(J)      AND VERTMISS(I,J)
            THEN CNT:=CNT+1;
            IF CNT < MIN THEN BEGIN X:=I; MIN:=CNT
                              END
      END;
      COVER := MIN;
      X
END MINVERTEX;
PROCEDURE FIN;

        COMMENT  TO CHECK WHETHER ALL THE VERTICES HAVE BEEN COLOURED
        IF YES THEN THE CHROMATIC NUMBER IS FOUND;

BEGIN LOGICAL FST; FST:=FALSE;
      FOR I:=1 UNTIL FULS+1 DO
      FOR J:=0 UNTIL 31 DO
      IF (¬STATE(NEW,I)) AND MASK SHL J = ZEROS THEN  IF ¬FST
                                                      THEN FST:=TRUE
                                                      ELSE GO TO XT;
      WRITE(   "CHROMATIC NUMBER OF GRAPH IS                  ",CHR);
      GO TO EXIT;
XT:
END FIN;
```

```
ZEROS:=#0; ONES:= ¬ZEROS; MASK:=#1;
TEMP:=ALLMISS;
FOR I :=1 UNTIL M-1  DO BEGIN MISS(I,1) := MISS1(TEMP);
                              MISS(I,2) := MISS2(TEMP);
                              TEMP := LINK(TEMP);
                         END;
MISS(M,1) := MISS1(TEMP); MISS(M,2) := MISS2(TEMP);
SN:= IF N>32 THEN 32 ELSE N;
FOR I:=1 UNTIL FULS DO STATE(1,I) := ONES;
STATE(1,FULS+1) := ONES SHR (32-XTRA);
PRESVERTEX(1,1) := ONES SHR (32-SN);
PRESVERTEX(1,2) := ONES SHR (64- N);
OLD:=NEW:=CHR:=1;
FIN;  NEW:=LU:=CHR:=2;
WHILE TRUE DO
BEGIN X := MINVERTEX;  J:=1;

      COMMENT  TO COLOUR ALL MISS M WHICH CONTAIN VERTEX X
               AND STORE THE MISS OF  G-M  IN STATE(NEW);

    WHILE COVER > 0 DO
    BEGIN IF STATMISS(J) AND VERTMISS(X,J) THEN
          BEGIN FOR  I:=1 UNTIL M DO TEMP1(I):=TEMP2(I):=FALSE;
                FOR I:=1 UNTIL FULS+1 DO STATE(NEW,I):=ZEROS;
                FOR Y:=1 UNTIL M DO
                IF STATMISS(Y) THEN
                IF INDEPENDENT(J,Y) THEN TEMP2(Y):=TRUE
                                    ELSE TEMP1(Y):=TRUE;
                FOR Y:=1 UNTIL M DO
                IF TEMP1(Y) AND ((Y>J) OR ¬VERTMISS(X,Y)) THEN
                BEGIN FOR Z:=1 UNTIL M DO
                      IF TEMP2(Z) AND SUBSET(Y,Z) THEN GO TO XT
                                                  ELSE
                      IF TEMP2(Z) AND SUBSET(Z,Y) THEN TEMP2(Z):=FALSE
                     ;TEMP2(Y) := TRUE;
             XT: END;
                FOR Y:=1 UNTIL M DO
                IF TEMP2(Y) THEN STATE(NEW,1+(Y-1)DIV 32) :=
                                 STATE(NEW,1+(Y-1)DIV 32) OR
                                 MASK SHL ((Y-1) REM 32);
                PRESVERTEX(NEW,1):=PRESVERTEX(OLD,1) AND ¬MISS(J,1);
                PRESVERTEX(NEW,2):=PRESVERTEX(OLD,2) AND ¬MISS(J,2);
                FIN;  NEW:=NEW+1; COVER:=COVER -1
          END;  J:=J+1
    END; OLD:=OLD+1; IF OLD=LU THEN BEGIN CHR:=CHR+1; LU:=NEW;
                                    END
END;
END;
EXIT:
END CHROMATIC;
```