A Graph Theory Model for the Computer Solution of University Time-tables and Related Problems

by

Michael Roy Williams

A thesis submitted to the University of Glasgow for the degree of Doctor of Philosophy.

Summary of Thesis.

"A Graph Theory Model for the Computer Solution of University Time-tables and Related Problems"

The work described in this thesis is concerned with four main fields of investigation, three concerned with the problems of a university administration in producing time-tables, and one concerne with the theory of graphs which provides a convenient mathematical model of a university's course-student structure.

A university administration's time-table problems may be classified under three headings:

1/ the production of examination time-tables,

2/ the assignment of students to classes, and

3/ the production of class-teacher-room time-tables.

These three problems are a class of the general combinatorial problem and thus simple enumeration will, in theory, provide a solution. This thesis describes and evaluates several algorithmic methods of solution and several heuristic approaches to reduce the combinatorial difficulties of the problems. Although heuristic methods do not guarantee the finding of an optimal solution, or, in some cases, any solution at all, the success of particular heuristic is demonstrated on actual course-student data.

A new algorithmic method is proposed for the construction of class-teacher-room time-tables. The feasibility of this method is demonstrated with a non-trivial example based on a game.

The thesis concludes with an investigation of the theory of graphs, the mathematical model used in previous work. Upper and lower bounds for the chromatic number of a graph are developed and procedures for reducing the size of the problem are constructed and discussed.

An algorithm for finding all the complete subgraphs of a graph is developed as an aid in determining the solution to parts of the time-table problem. This is then related to several theorems concerning the eigenvalues and eigenvectors of the matrices associated with graphs and their meaning in the terms of the structure of these graphs. This leads readily to a bound, involving eigenvalues, for the size of the largest complete subgraph in any given graph.

The graph theory section ends with a short note on the four colour problem.

Acknowledgements

I should like to express my thanks to my colleagues in the Computing Department at the University of Glasgow for their help and encouragement in preparing this thesis, and the many stimulating discussions which have helped me to modify and extend my original ideas. My thanks are also due to Professor J. E. L. Peck, who originally suggested the topic, and to the Office of the Registrar, University of Alberta, Calgary who very kindly supplied me with two complete sets of student data, without which I could not possibly have progressed as far as I have.

Finally a special word of thanks to Professor D. C. Gilles for his support, encouragement, and the suggestions he has given me, and the University of Glasgow for providing the opportunity for me to do this work.

M. R. Williams

Table of Contents

I N T R O D U C T I O N

Education is very big business. Next to national defence it is probably the largest single objective in every civilized country. It intimately concerns about 20% of the population and is one of the major items in the national budget. With millions of people and vast amounts of money to be accounted for, educational administration is ripe for the application of computers to some of the difficult problems arising in educational institutions.

A very large curriculum reform is in progress in response to the expansion of knowledge and the general dissatisfaction with the school programs of the 1930's and 1940's. There are now many groups working in different countries studying changes needed in the school curriculum. Each of these groups is producing the hundreds of items necessary for an instructional package. But underlying each package is an assumption of how the curriculum should be organized. To date the computer has played a very minor role in this curriculum reform. However as the reform movement grows towards more and more individualized instruction, the computer will become not only a great help but the only possible means for a large institution to deal with its curriculum problems.

This thesis considers three aspects of the workload of an educational administration:

1/ the production of examination time-tables

2/ the assignment of students to classes

3/ the production of class-teacher-room time-tables

These three problems are a class of the general combinatorial problem. The fact that complete enumeration of all possible assignments or time-tables provides a theoretically satisfactory solution to the problem is evident, however the practical impossibility of applying total enumeration to any but the smallest of problems is also quite clear. An experienced person is able to produce a reasonable time-table, reasonable, that is, in the effort required to find a better one, because he can see to avoid the many unfruitful paths the computer would have to take. However as universities increase in size and complexity the effort involved in this task will grow to the point where computer based methods may prove to be the only reasonable method of accomplishing them.

This thesis describes and evaluates several algorithmic methods of solution and several heuristic approaches to eliminate the combinatorial impossibility of the problems. Although the use of heuristics eliminates the assurance of eventually obtaining a solution (if, indeed, one exists at all) and will not guarantee that, if found, the solution is optimum, the success of particular heuristics will be demonstrated.

The computational power of an English-Electric-Leo-Marconi KDF 9 computer was used to develop and check out the procedures. The main part of this thesis is a description of the ideas leading to the development of the procedures, however some of the approaches found in the literature have been included for reasons of completeness.

Although the organization of schools differs from area to area the basic procedures are applicable to all levels of school organization. The emphasis has been placed on a situation similar to that of a North American university as it is somewhat more general than either a school or a British university situation, and the author was very familiar with a North American university administration. The fact that many problems, not directly related to the production of time-tables, may be expressed in the same notation as that used in this thesis gives an indication that these procedures may have wider applicability than just educational administration.

It is hoped that, by an examination of these and other procedures, both heuristic and algorithmic, someone may be able to determine a relationship between the data and the time-tables produced. As J. Von Neumann (16) once said:

> "That the first, and occasionally the most important, heuristic pointers for new mathematical advances should originate in

> physics (experimentation) is not a new or surprising occurrence. The calculus itself originated in physics. The great advances in the theory of elliptic differential equations originated in equivalent insights. This applies even in the heuristic approach to the correct formulation of their uniqueness theorems and of their natural boundary conditions."

Thus by expounding heuristic methods perhaps an insight may be obtained into an area where modern mathematics can not go, just as it once could not delve into the inner mysteries of elliptic differential equations.

The rest of this work is divided into four major chapters. The first chapter deals with the production of examination time-tables. After developing a graph theoretic model, an heuristic procedure for the production of the time-tables is developed. A number of other authors have developed very similer heuristics for producing examination time-tables, however it appears that all the authors have worked in ignorance of one anothers work. Using this heuristic as a base it is then possible to show the relevance and use of an eigenvector of one of the matrices used. This leads to an improvement in the basic heuristic resulting in an extremely good procedure. In order to show the success of the improved heuristic an investigation of possible algorithmic procedures is conducted, resulting in the development of an algorithm for finding complete subgraphs of a given graph. This procedure is partially based on Theorem 1.5.3, the statement of which, but not the proof, is attributable to Dr. A. R. Meetham from the National Physical Laboratory. The chapter ends with a brief summary of the computational results and a note on the possible modifications to the procedure.

Chapter 2 deals with the problem of assigning students to classes. After showing the relevance of the problem, an heuristic procedure is developed and compared to those previously described in the literature. A section is then devoted to describing the solution to the problem in terms of transportation networks. This section is an extension of a general work by Ford and Fulkerston (41). The chapter finishes with the development of an algorithmic sectioning procedure, based on the complete graph algorithm, and a discussion of some aspects of its implementation.

Chapter 3 is a review of some of the literature dealing with the problem of producing full master time-tables and a discussion of an algorithmic procedure. The chapter concludes with an example, based on a game, of how the procedure would operate.

Chapter 4 investigates the theory behind the problems. Upper and lower bounds for the chromatic number of a graph are developed and procedures for reducing the size of the problem are developed and discussed. A section is devoted to the eigenvalues and eigenvectors of various matrices and their association with the colouring problem. The chapter finishes with a justification for the heuristic procedure developed in Chapter 1 and a short note on the four colour problem.

C H A P T E R 1

Examination Time-tables

Section 1.1 The Problem

The character of the examination time-table problem allows it to be readily represented by a mathematical model known as a graph (2,4,19). A graph is:

1/ a set X

2/ a function U mapping X into X.

Or, to put it another way, a graph G, which is denoted by

$$G = (X,U),$$

is the pair consisting of the set X and the function U. It is convenient to visualize the set X as points or vertices in a plane, and if x and y are two vertices such that

$$y \in Ux \qquad \text{and} \qquad x \in Uy,$$

then the two vertices will be joined by a line or edge. If x and y are two vertices such that

$$y \in Ux \qquad \text{but} \qquad x \notin Uy$$

then x and y will be joined by an edge oriented in the direction y to x.

Graphs are met with in different disciplines under different names: in psychology they are called sociograms; in topology, simplexes; in physics and engineering, circuit diagrams. In the context of the production of an examination time-table the set of vertices, X, will represent the set of classes offered at the educational institution and the function U will be such that if any student is taking both course x and course y then U will generate an undirected edge between vertex x and vertex y. For example in FIGURE 1.1.1 are listed the courses being taken by four students and the graph generated by this data.

The general problem of producing examination time-tables is one of partitioning the vertices of these graphs into independent or disjoint sets, such that each set contains no pair of vertices which are connected by an edge. This may be considered as "colouring" the vertices of the graph. A colouring of a graph, using at most k colours, is a function C defined over the vertices of the graph and taking one of the values $1,2,3,\ldots,k$ at each vertex with the condition that

$$C(x) \neq C(y)$$

if the vertices x and y are joined by an edge. If the graph is colourable in k colours but not with k - 1 colours then k is called the chromatic number and the graph is said to be k chromatic. The symbol

$$\gamma(G)$$

Student A	Student B	Student C	Student D
course 1	course 4	course 1	course 2
course 2	course 5	course 4	course 5
course 3	course 6	course 3	course 3

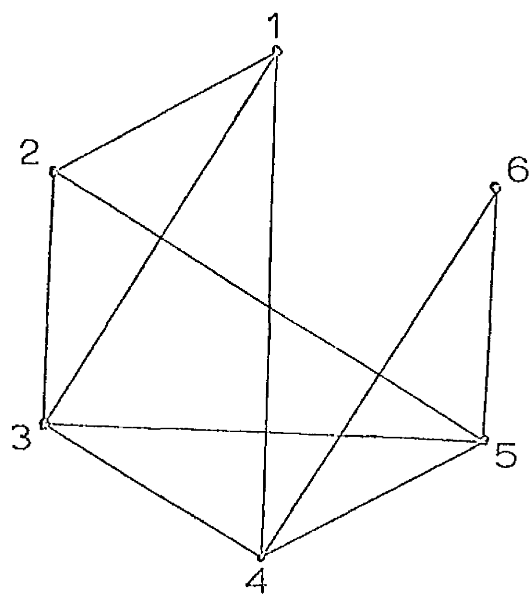


FIGURE 1.1.1

Showing the courses taken by four students and the graph generated by this data.

will denote the chromatic number of the graph G.

If a graph consists of two or more disconnected components the chromatic number of the whole graph is that of the component with largest chromatic number.

Section 1.2 Methods of Solution

To find the chromatic number and the colours assigned to the vertices of a graph G (with N vertices and E edges) it is possible to use an empirical procedure which is straightforward and capable of direct implementation, but not always effective, an analytic procedure which gives a solution systematically but requires a tremendous amount of computation, or an heuristic procedure which, although it does not guarantee an optimum solution, can in practice give an acceptable answer with a minimum of effort.

The empirical procedure consists of starting with an arbitrary colouring, using the colours $1,2,\ldots,p$ and attempting step by step to eliminate one of them. This can be readily seen to be an awkward and not necessarily successful procedure if implemented on a large complex graph.

The analytic procedure consists of testing analytically whether the graph can be coloured with p colours. With any scheme using p colours it is possible to associate numbers $S(i,j)$ and $C(i,q)$ (where $i = 1,2,\ldots N$; $j = 1,2,\ldots,E$; $q = 1,2,\ldots,p$) such that:

$$C(i,q) = \begin{cases} 1 & \text{if vertex } i \text{ is of colour } q \\ 0 & \text{otherwise} \end{cases}$$

$$S(i,j) = \begin{cases} 1 \text{ if edge } j \text{ is incident with vertex } i \\ 0 \text{ otherwise} \end{cases}$$

The problem of determining if the graph can be coloured in p colours now reduces to finding integers C(i,q) such that

$$C(i,q) \geq 0 \quad (i = 1,2,\ldots,N;\ q = 1,2,\ldots,p)$$

$$\sum_{q=1}^{p} C(i,q) = 1 \quad (i = 1,2,\ldots,N)$$

$$\sum_{k=1}^{N} S(k,j)C(k,q) = 1 \quad (j=1,2,\ldots,E;\ q=1,2,\ldots,p)$$

Thus there exists a system of linear inequalities whose compatibility may be investigated by the usual methods of integer programming. If integers C(i,q) can be found satisfying the above constraints then p may be systematically reduced until the chromatic number, and thus the values of the colouring function, are determined.

Unfortunately the analytic procedure also breaks down on large graphs because of the rapidly increasing computation necessary as the size of the graph gets larger. The computation effort may be reduced if the graph is separable into several disjoint subgraphs.

The individual connected subgraphs may be easily determined by considering the original data and not the graph. Each student will have taken a set of $|R|$ courses, R, which is a subset of the N courses offered by the institution. By considering each set R in turn the following algorithm will easily determine the connected subgraphs:

1/Produce an N element vector B such that

$$B_i = i \quad (i = 1,2,3,\ldots,N)$$

2/Select a course j from R such that

$$R_j = \min.\ R_k \quad (k = 1,2,3,\ldots,|R|)$$

3/For each R_k $(k = 1,2,3,\ldots,|R|)$ replace each occurrence of the number R_k in B by B_j

4/After considering all the sets R, the vector B is scanned and if

$$B_i = B_j$$

then i and j are vertices in the same connected subgraph.

The integer programming procedure may now be applied to each disconnected subgraph in turn. If the graph is not separable, or if each disconnected subgraph is still too large to make an analytic procedure practicable, then recourse may be made to finding a "point of articulation" if one exists. A point of articulation is a vertex, p, which separates the vertices of the graph into two or more subsets, V_1, $V_2,\ldots,V_n$, having only p in common and such that any edge chain between a vertex in V_i and a vertex in V_j must pass through p. For example the graph in FIGURE 1.2.1 vertices c and d are points of articulation.

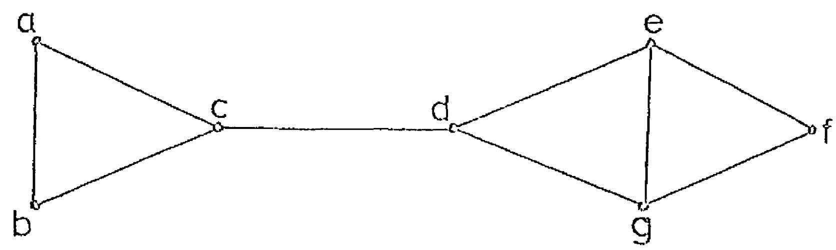


FIGURE 1.2.1

Showing a graph in which vertices c and d are points of articulation.

Removal of vertex c separates the graph into two disconnected subgraphs. If the subgraphs are now coloured it will only be necessary to assign a colour to vertex c such that (perhaps after permuting the colours of one of the subgraphs) it is different from the colours assigned to vertices a, b, and d.

This concept may be extended to finding a "minimal articulated set". This is a set (not necessarily unique) consisting of the least number of vertices whose removal will divide the graph into two or more unconnected subgraphs. The problem of finding a minimal articulated set is not trivial.

To find the minimal articulated set of a graph, $G = (N,U)$, N must be divided into three subsets N1, N2, and a such that:

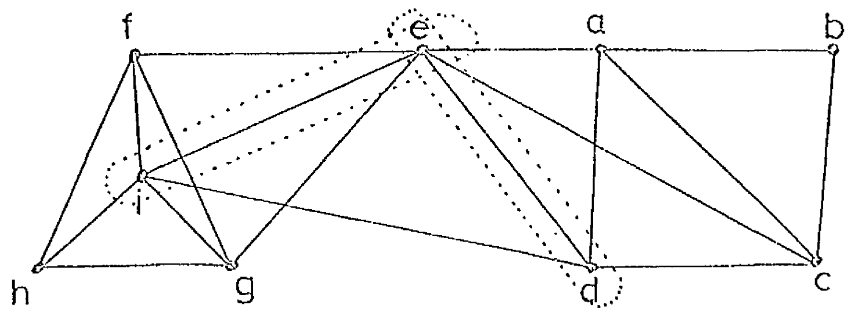
$$U_{N1} \wedge N2 = \emptyset \qquad (1.2.1)$$

$$U_{N2} \wedge N1 = \emptyset \qquad (1.2.2)$$

$|a|$ is minimal.

Let M be the boolean matrix such that

$$M_{ij} = \begin{cases} 1 & \text{if vertex } i \text{ is adjacent to vertex } j \\ 0 & \text{otherwise} \end{cases}$$



$$M = \begin{array}{c|ccccccccc}
 & a & b & c & d & e & f & g & h & i \\ \hline
a & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
b & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
c & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
d & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
e & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
f & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
g & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
h & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
i & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0
\end{array}$$

$$\bar{M} = \begin{array}{c|ccccccccc}
 & a & b & c & d & e & f & g & h & i \\ \hline
a & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
b & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
c & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
d & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
e & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
f & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
g & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
h & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
i & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1
\end{array}$$

<u>FIGURE 1.2.2</u>

Showing a graph, its matrix M, the complement matrix $\bar{M}$, and two minimal articulated sets

FIGURE 1.2.2 shows a graph, its matrix M and the complement $\bar{M}$ ($\bar{M}$ = 1-M) of M. The submatrix of $\bar{M}$ defined by the rows corresponding to N1 and the columns corresponding to N2 has all its elements equal to 1, as defined by the relations (1.2.1) and (1.2.2) above. The problem thus reduces to finding the largest complete (ie. all elements equal to 1) submatrix of $\bar{M}$. Kaufmann (23) describes an algorithm which will yield both

1/ N1 = a,b,c,d   N2 = f,g,h

a = e,i   and |a| = 2

and

2/ N1 = a,b,c   N2 = f,g,h,i

a = d,e   and |a| = 2

Unfortunately Kaufmann's algorithm is quite expensive in computer time and, as the size of the graph increases, more and more minimal articulated sets will be needed, thus making it almost impossible to ensure that the colourings of the individual subgraphs will be compatible. Because of these difficulties the analytic procedure must be counted as impracticable for large graphs.

An heuristic procedure offers neither the assurance of finding an optimal solution, as does the analytic procedure, nor the simplicity of the empirical procedure, but it does offer the ability to obtain a solution in a practical case and to obtain this solution without an unreasonable amount of computation.

Section 1.3 Heuristic Procedures

The author (28), A. J. Cole (5) and others have proposed heuristics for solving this problem. The author's work (hereafter refered to as the Peck-Williams procedure) is slightly more general than most of the others but they all follow the same general pattern. The main heuristic assumption is: If a course (vertex) i conflicts with a large number of other courses, then it will be harder to find a time period (colour) to fit it in than to find a time period for a course j which conflicts with only a few other courses.

If d_i denotes the degree of vertex i (the number of edges incident with vertex i) then this becomes an index of the extent to which course i conflicts with other courses (for example in FIGURE 1.3.1, $d_3 = 4$ which indicates that course 3 is in direct conflict with three other courses, plus one because $a_{33} = 1$).

Assuming that the first T-1 periods of an examination time-table are complete, then to select the courses which will write examinations in period T the heuristic procedure would be as follows:

1/find the unassigned course with the largest d_i

2/check to see if this course is joined by an edge to any other course already assigned to period T; if no edge exists then assign this course to period T and go to step 1, otherwise remove this course from further consideration in period T and go to step 1.

This heuristic has two distinct advantages:

1/it has intuitive appeal

2/it is very simple to implement on even very small computers.

The intuitive appeal stems from the experience of "hand" produced examination time-tables, where to fit a new course, having a large student population (and hence a large number of conflicts with other courses) into an already completed time-table is an almost impossible task. It is for this reason that a clerk of examinations will always time-table the large classes first and then let the classes with a small population fit in where they can.

The Peck-Williams procedure has been successfully implemented on a very small I.B.M. 1620 where the core store was not sufficient to hold all the required information. Resort was made to a large loop of paper tape which was searched to find the items of information that were required. Even under such a severe handicap the procedure produced usable results in a realistically short time. The procedure proposed by Cole has been implemented on an Elliot 803 and, although limited to less than 340 subjects, also produced usable results without using vast amounts of computer time.

For greater generality consider a graph $G = (V,U)$ whose vertices are $v_1, v_2, \ldots, v_n$. Let a_{ij} be the number of edges of G going from vertex v_i to vertex v_j. The square matrix A with n rows and n columns is called the matrix associated with the graph G. The element a_{ii} is meaningless in the context of examination time-tables. It will be a convention that $a_{ii} = 1$ unless otherwise stated. In most situations the elements of A will only take the values 0 or 1, in this case A may be considered to be a boolean matrix with $0 \equiv$ _false_ and $1 \equiv$ _true_. It will often be convenient to consider A both as a numeric matrix suitable for computation and as a boolean matrix for use in logical operations. The context of the argument will make clear which form of A is being used.

The heuristics, in the above mentioned procedures, actually operate on the associated matrix of the graph of the course conflicts. The ordering criterion, d_i, is obtained by

$$d_i = \sum_j a_{ij}$$

and the adjacency of two vertices, v_i and v_j, may be determined by inspection of the element a_{ij}.

By this heuristic the first course scheduled to hold its examination in period 1 will be the course whose vertex has the largest degree. However complications arise when two or more courses have vertices of the same degree. It is obvious that the selection of course i, for inclusion in period T, may produce a significantly different time-table from the one produced if course j had been initially selected.

The Cole procedure differentiates between courses of equal d_i by selecting the subject with the largest number of multiple papers which must be written on consecutive days. If no course emerges unique from this criterion then a selection is made by considering the number of papers written in each subject. If this still does not yield a unique course for consideration then the original ordering of the courses is considered and the first course encountered, meeting all of the above conditions, is selected for inclusion in time period T.

The Peck-Williams procedure, on the other hand, simply selects the first course it encounters with the appropiate d_i.

It is interesting to note that Holzman and Turkes (22) in one of the most widely read reports on this subject, while considering the order of scheduling classes, state:

> "an arbitrary policy states that the variables should be scheduled in the order $A_3, A_2, A_5, \ldots$".

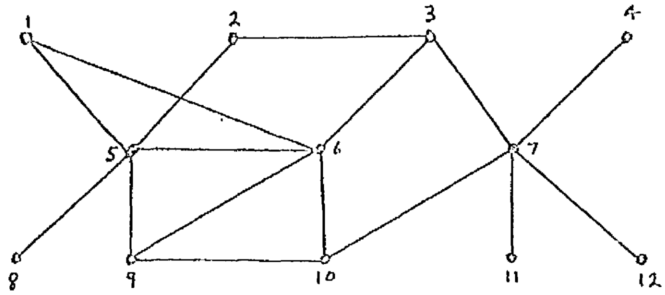
They go on to develop this arbitrary policy into a procedure which does not do justice to the word optimal which appears in the title of their report. By claiming to rely heavily on Bellman's principle of optimality, which states:

> An optimal policy has the property that, whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with respect to the state resulting from the first decisions,

they have produced a procedure which, although very cunning in the way it adds a course to the partially completed time period, still violates Bellman's principle, with respect to the whole time-table, by selecting the initial courses for each period according to an arbitrary policy.

This lack of a decision criterion for the selection of courses is a serious drawback of all these procedures as it can be shown that an incorrect choice of vertex can lead to time-tables which are far from optimal. For example, consider a selection of twelve courses whose conflict pattern produces the graph and its associated matrix shown in FIGURE 1.3.1. There is no single vertex of maximum degree, rather the choice lies between vertices 5, 6, and 7, each of which has a degree of six.

Both the Cole and the Peck-Williams procedure would have choosen vertex five as the initial assignment and thus produced a time-table of four periods as



$$
A = \begin{array}{cccccccccccc c}
 & & & & & & & & & & & & d_i \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 4 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 6 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 6 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 6 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 2
\end{array}
$$

FIGURE 1.3.1

Showing a graph, its associated matrix and the degree of each vertex.

follows:

	period 1	period 2	period 3	period 4
	5	6	3	10
	7	2	9	
<u>courses</u>		4	1	
		8		
		11		
		12		

An inspection of this trivial graph will reveal that its chromatic number is, in fact, three and it should be possible to produce an examination time-table of the form:

period 1	period 2	period 3
6	5	9
7	10	1
2	3	
8	4	
	11	
	12	

Thus the ordering criterion used in these heuristic procedures is not optimal and should, if possible, be changed.

Section 1.4 An Improvement

Section 1.4 An Improvement

In investigating the required changes to the heuristic procedure it will be useful to define the "influence" or "degree of order k" (d_i^k) of a vertex. This influence will be an index of the degree of the vertices joined, by an edge path of length k, to the vertex under consideration.

For example, consider a tournament with four players v_1, v_2, v_3, v_4; if v_i defeats v_j then v_i is joined to v_j by two edges directed from v_i to v_j, if the match was drawn then the two vertices are again joined by two edges - one directed towards each. At the end of the tournament the results are placed into the graph and its associated matrix shown in FIGURE 1.4.1 (the loops on each vertex mean simply that each player is only as strong as himself).

The term a_{ij}^k is the general element of the matrix A^k (ie. the number of edge paths of length k between the vertices v_i and v_j) and

$$d_i^k = \sum_j a_{ij}^k \ .$$

Thus d_i^1 is the degree of vertex v_i (ie. $d_i^1 = d_i$).

In the example of the tournament in FIGURE 1.4.1

$$d_1^1 = 5 \qquad d_2^1 = 3$$

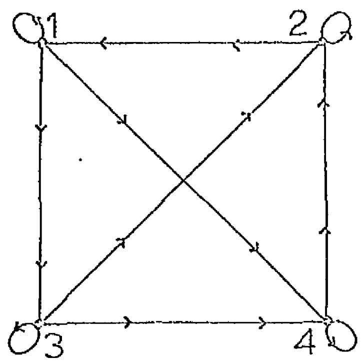
$$d_3^1 = 5 \qquad d_4^1 = 3.$$

v_1 defeats v_3 and v_4

v_2 defeats v_1

v_3 defeats v_2 and v_4

v_4 defeats v_2



$$A = \begin{matrix} 1 & 0 & 2 & 2 \\ 2 & 1 & 0 & 0 \\ 0 & 2 & 1 & 2 \\ 0 & 2 & 0 & 1 \end{matrix}$$

FIGURE 1.4.1

Showing the results of a tournament, their transformation into a graph, and the matrix associated with the graph.

Contestants v_1 and v_3 both have a degree of 5, while contestants v_2 and v_4 have a degree of 3. So no one player has seemingly emerged victorious. Continuing the process by finding d_i^2, where

$$d_i^{n+1} = \sum_j a_{ij} d_j^n \qquad (1.4.1)$$

the following results are obtained:

$$d_1^2 = 21 \qquad d_2^2 = 13$$

$$d_3^2 = 17 \qquad d_4^2 = 9.$$

This indicates that player v_1 is the winner of the tournament. This is due to the fact that the players defeated by v_1 (v_3 and v_4) were stronger than those defeated by v_3 (v_2 and v_4).

It has been suggested by Berge (2), in a discussion on tournament theory, that the "iterated power of order k of the vertex v_i" (π_i^k) be defined as

$$\pi_i^k = \frac{d_i^k}{\sum_i d_i^k}$$

It is well known that the $\lim_{k\to\infty} \pi_i^k$ exists for positive matrices, and the vector:

$$\pi^k = (\pi_1^k, \pi_2^k, \cdots, \pi_n^k)$$

tends toward the eigenvector corresponding to the largest real positive eigenvalue of the matrix A.

A brief look at a completely different problem may provide a further insight into the nature of the quantity d_i^k.

In an attempt to devise a scheme for drawing a graph on a computer controlled plotter or display it on a cathode ray device, the following problem arose:

> given n vertices, some of which are joined together by edges, produce a pair of X,Y co-ordinates for each vertex such that when the vertices are distributed on their co-ordinates the sum of the distances between the bound pairs of vertices is minimal, the centre of gravity of the system is on the origin and the whole system is distributed evenly over a circular display area.

If the co-ordinates of vertex i are X_i and Y_i then

$$\sum_i X_i = 0 \quad \text{and} \quad \sum_i Y_i = 0 \qquad (1.4.2)$$

to keep the plot centred on the origin and

$$\sum_i X_i^2 = c \quad \text{and} \quad \sum_i Y_i^2 = c \qquad (1.4.3)$$

to contain the plot within a constant area. If the matrix A is the n by n symmetric zero-one matrix associated with the given graph then a function, f, may be constructed which will provide an iterative basis for the assignment of an X and Y to each vertex.

If

$$f = \frac{\text{sum of the squares of the distances between all joined vertices}}{2}$$

then

$$2f = \sum_i \sum_j ((X_i - X_j)^2 + (Y_i - Y_j)^2)\, a_{ij} \qquad (1.4.4)$$

Expanding this it is possible to obtain

$$f = \sum_i (X_i^2 \sum_j a_{ij}) - 2\sum_j \sum_i X_i X_j a_{ij} + \sum_j X_j^2 \sum_i a_{ij}$$
$$+ \sum_i (Y_i^2 \sum_j a_{ij}) - 2\sum_j \sum_i Y_i Y_j a_{ij} + \sum_j Y^2 \sum_i a_{ij} \qquad (1.4.5$$

To produce the best clustering the function f must be a minimum. Considering one vertex, i, and for simplicity assuming that the graph has no loops, ie. $a_{ii} = 0$, it is possible to obtain

$$\frac{\partial f}{\partial X_i} = X_i \sum_j a_{ij} - \sum_j X_j a_{ij} \qquad (1.4.6)$$

Setting $\frac{\partial f}{\partial X_i} = 0$ and solving for X_i it is found that

$$X_i = \frac{\sum_j X_j a_{ij}}{\sum_j a_{ij}}; \qquad (1.4.7)$$

similarly

$$Y_i = \frac{\sum_j Y_j a_{ij}}{\sum_j a_{ij}} \qquad (1.4.8)$$

By using (1.4.7) and (1.4.8) in an iterative procedure it is possible to determine the X_i and Y_i for each vertex. Obviously, in the practical display problem, it is necessary to scale X_i Y_i and shift the origin between each iteration to satisfy conditions (1.4.2) and (1.4.3) and extra steps must be taken to ensure that closely related groups of vertices do not shrink to a single point or all vertices come to cluster along the line $X = Y$.

The similarity between (1.4.7) and (1.4.1) is striking but not unexpected, for in both problems the object is to find the "centre of gravity" of the graph. In (1.4.7) the denominator may be interpreted as a factor tending to pull a vertex of high degree to the centre of the system.

In attempting to assign colours to the vertices of a graph it will be the vertex with the largest d_i^k (from (1.4.1)) or the smallest X_i (from (1.4.7)) that is most likely to cause trouble as it is the vertex most deeply embedded in the system.

Returning to the problem of examination time-tables; it should now be clear that, because $\lim_{k \to \infty} \pi^k$ tends toward the principal eigenvector corresponding to the largest eigenvalue of the matrix associated with the graph of their conflicts, this eigenvector should be used as the ordering criterion. In any practical situation the computation of this eigenvector is difficult (and on small computers its computation would be prohibitive), thus the courses should be ordered by their d_i^k where k is large enough to obtain sufficient separation of the classes to make the order of scheduling clear. The actual value of k that should be used will vary with the size of the problem and as the nature of the graph. In general the iterative procedure should be carried out to as high a k as possible, notwithstanding the fact that if a clear separation of the vertices is obtained (no two elements of d^k being equal) then the iterative procedure should be stopped. It should be noted that, as in the case of v_{11} and v_{12} in FIGURE 1.3.1, a complete separation may never be obtained irrespective of the number of iterations performed.

Returning to FIGURE 1.3.1 to consider a concrete example, the associated matrix and the first three d_i^k's are

												d_i^1	d_i^2	d_i^3
1	0	0	0	1	1	0	0	0	0	0	0	3	15	66
0	1	1	0	1	0	0	0	0	0	0	0	3	13	56
0	1	1	0	0	1	1	0	0	0	0	0	4	19	79
0	0	0	1	0	0	1	0	0	0	0	0	2	8	28
1	1	0	0	1	1	0	1	1	0	0	0	6	24	107
1	0	1	0	1	1	0	0	1	1	0	0	6	27	125
0	0	1	1	0	0	1	0	0	1	1	1	6	20	83
0	0	0	0	1	0	0	1	0	0	0	0	2	8	32
0	0	0	0	1	1	0	0	1	1	0	0	4	20	91
0	0	0	0	0	1	1	0	1	1	0	0	4	20	87
0	0	0	0	0	0	1	0	0	0	1	0	2	8	28
0	0	0	0	0	0	1	0	0	0	0	1	2	8	28

Using d_i^1 as the ordering criterion the Peck-Williams procedure produces an examination time-table of four periods as follows:

period 1	period 2	period 3	period 4
5	6	3	10
7	2	9	
	4	1	
	8		
	11		
	12		

If this procedure is changed to use d_i^3 as the ordering criterion it is possible to produce a three period time-table:

period 1	period 2	period 3
6	5	9
7	10	1
2	3	
8	4	
	11	
	12	

This is the best possible in this case. Thus with an expenditure of a small effort in computing d^k a significant improvement can be made in the heuristic even for trivial graphs.

Section 1.5 Determination of the Chromatic Number

In determining an examination time-table by heuristic methods it can not be assumed that the result is optimal, or indeed anywhere near optimal. For this reason a great deal of time was spent on methods of checking the result to see if it could be further improved.

The mathematical basis of graph theory has not progressed to the point where, given a graph, a formula may easily be determined to give the chromatic number, much less indicate which vertices should be given what colours.

One approach to the determination of the chromatic number of a graph may be made through the theory of chromatic polynomials. A chromatic polynomial is a function, $F(\lambda)$, which expresses the number of different ways of colouring a graph as a function of the number of colours used, λ. For example, in FIGURE 1.5.1 the centre vertex may be coloured in any of the λ colours, the two outer vertices may now be coloured independently each in $\lambda - 1$ ways. Thus

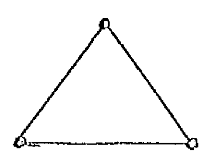
$$F(\lambda) = \lambda(\lambda-1)^2.$$

Similarly in FIGURE 1.5.2 the top vertex may be coloured in λ ways, there are then $\lambda-1$ ways of assigning a colour to one of the adjacent vertices and $\lambda-2$ ways

$$F(\lambda) = \lambda(\lambda-1)^2$$

FIGURE 1.5.1

Showing a graph and its chromatic polynomial



$$F(\lambda) = \lambda(\lambda-1)(\lambda-2)$$

FIGURE 1.5.2

Showing a graph and its chromatic polynomial of colouring the third vertex. Thus

$$F(\lambda) = \lambda(\lambda-1)(\lambda-2).$$

It has been shown by R. C. Read (30) that $F(\lambda)$ is always a polynomial of the following form:

$$F(\lambda) = C_r\lambda^n - C_{r-1}\lambda^{n-1} \,\ldots\ldots\ldots\, C_{r-n}\lambda$$

where n is the number of vertices in the graph,

$$C_r = +1,$$

$$C_{r-1} = \text{the number of edges in the graph,}$$

and the sign of C_i alternates at each term.

If $F(\lambda)$ is the number of ways of colouring the graph in λ colours then the smallest positive integer (excluding zero), λ, such that

$$F(\lambda) \geq 0$$

will be the chromatic number of the graph.

The computation of the coefficients of the polynomial is, in general, an impossibly complex process for a large graph. In fact very little is known about chromatic polynomials and only one way has ever been found to construct them. A number of theorems in the paper by Read (30) give necessary conditions for a polynomial to be the chromatic polynomial of some graph, but none of them give a sufficient condition.

Because of the various limitations of these and other methods, it became necessary to retreat to basic concepts and attempt to design a procedure which would give some information on the chromatic number of a graph.

J. De Bruijn (unpublished but his proof appears in a paper by G. A. Dirac (8)) has shown that a graph always contains a critical chromatic subgraph and that this subgraph is finite and connected (a critical graph is one in which if you delete an arbitrary vertex or edge you reduce the chromatic number of the graph). This result was improved by Brooks (8) who showed that if $k \geq 4$ a critical k-chromatic graph contains either a k-complete graph or a vertex of degree k. Dirac (9) was then able to show that, if $0 \leq p \leq k-1$, a critical k-chromatic graph contains either a complete k-p graph as a subgraph or has at least k+p+2 vertices. Dirac (10) was then also able to show that if a critical k-chromatic graph contains $n < k$ vertices (ie. it is not a complete graph on k vertices) and e edges then the relation

$$2e \geq (k-1)n+k-3$$

must hold true.

With the above results in mind, an attempt was made to investigate the properties of the graph (defined by its associated matrix) of each of the data sets used in the examination time-table experiments. In order to examine the colouring properties of a graph it would be useful to find the size and composition of the critical chromatic subgraph. However the possible complex nature of its construction, as indicated by the example in FIGURE 1.5.3 (a critical chromatic graph of 8 vertices whose $\gamma(G) = 6$ yet the largest complete subgraph is of order 5), rules out any reasonable method of determining it exactly.

It is difficult (though not impossible) to deliberately construct a critical k-chromatic graph which does not contain a complete graph of order k, k-1, or k-2. It is therefore reasonable to suppose that, in the graphical problems arising out of the physical world, the critical chromatic graphs will be composed of either a complete graph of order k, or a complete graph of order k-p, where p will be a small (with respect to k) integer. Thus an algorithm for determining the size and composition of the largest complete graph, inbedded in the course conflict graph, was the prime objective of the investigation. A complete graph of order n will be denoted by K_n.

A number of attempts at this problem have been found in the literature, most of which were by sociologists attempting to analyze cliques or other group structures in sociograms. Typical of the approaches was that used by Forsyth and Katz (15). They used the following empirical procedures:

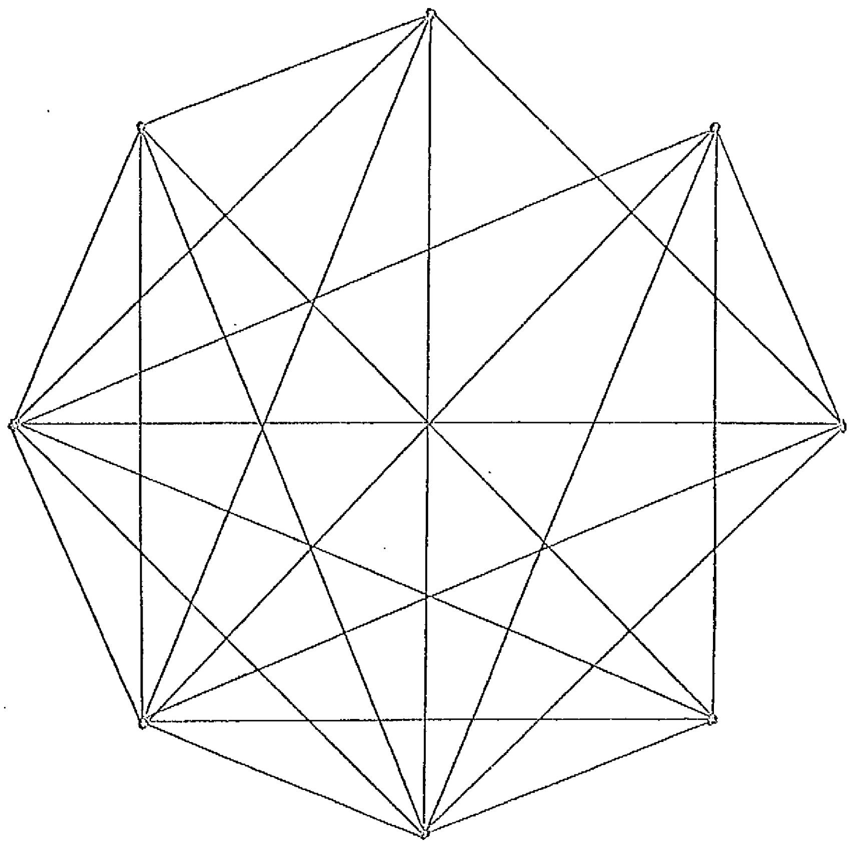


FIGURE 1.5.3

A graph requiring six colours; ~~whose~~ the largest complete subgraph is of order five.

1/choose a vertex, a

2/ehumerate all vertices connected to a

3/interchange rows and columns of the associated matrix so that the rows and columns, correspondi to the vertices found in step 2, are side by side

4/interchange the rows and columns of this submatrix until as many of the non-zero elements as possible are clustered near the diagonal

5/the largest submatrix whoes elements are all non-zero corresponds to the largest complete graph.

This rather unwieldy procedure was eventually replaced by one developed by Harrary and Ross (20). They made use of the fact that, given a graph G and its associated matrix A, the powers of A yield a matrix whose elements a^p_{ij} (the i,j th element of the matrix $A \times A \times A$ .... p times) are the number of paths of length p going from vertex i to vertex j. This leads readily to the fact that each element in the diagonal of the cube of the associated matrix of a K_n is the number

$$(n-1)(n-2)$$

where n is the number of vertices in the complete graph. Thus by cubing an associated matrix and inspecting the diagonal it is possible to determine the largest complete graph embedded in the system.

This method is limited to those cases where there exists only one (or perhaps several disjoint) complete graph and can not be made applicable to a graph, such as that in FIGURE 1.5.4, made up of several interconnected complete graphs.

In attempting to remedy these faults it was noted that a K_n is made up of a series of K_2s and, if $n \geq 3$, then it is made up of a series of K_3s. In general the following results may be obtained.

Theorem 1.5.1

A complete graph, $G(X,U)$, of order n $(n \geq 3)$, will contain, as subgraphs, n complete graphs of order $n-1$.

Proof

Delete one vertex, r, from the graph G along with the edges such that $r \in U_x$ (for all $x \in X-r$). There are now $n-1$ vertices left, and these are connected by edges such that $x \in U_y$ (for all $x,y \in X-r$). This, by definition, is a complete graph of order $n-1$ on the $n-1$ vertices in the set $X-r$. As r may be any of the n vertices in X there must be n complete graphs of order $n-1$ in G.

A slightly more general result is:

Theorem 1.5.2

A complete graph, $G(X,U)$, of order n ($n \geq 3$) contains, as subgraphs, $n!/(p!(n-p)!)$ complete graphs of order p ($2 \leq p \leq n$).

Proof

Delete any $n-p$ vertices along with any edges incident with them. By the same argument used in Theorem 1.5.1, the remaining graph is a complete graph of order p. As the $n-p$ vertices, deleted above, may be any vertices in the set X, it is obvious that the number of complete graphs of order p as subgraphs of G is the number of combinations of n things taken p at a time, or $n!/(p!(n-p)!)$.

Theorem 1.5.3

In a complete graph $G(X,U)$, of order n, ($n \geq 3$), each edge is part of $n-2$ edge circuits of length three.

Proof

Consider an edge, α, between the vertices i and j. If any other vertex r ($r \in X-i-j$) is taken then, by the definition of a complete graph, $r \in Ui$ and $r \in Uj$ and therefore an edge circuit of length three exists and consists of the edges α, U_{jr}, and U_{ir}. As r may be any of the $n-2$ vertices in $X-i-j$ the theorem is proved.

By the results deduced above it should be possible to design an algorithm to find all the K_ns then see if the vertices of these graphs form a K_{n+1} and continue in this manner until the largest complete graph is found.

Because of the combinatorial nature of the problem it would be helpful if the search could be started with n as large as possible, and yet still be certain that a K_n still existed. This would eliminate part of the very time consuming search.

P. Erdös (11) has shown that if a graph G, with n vertices, has

$$\frac{n^2}{4} + h$$

edges then it contains at least

$$\frac{n}{2} + h - 1$$

complete three graphs, if it contains any at all. This, although it looks as if it should provide a starting point for an iterative procedure, proves useless because

$$\frac{n^2}{4}$$

is such a large number. For example, one of the data sets used in this investigation gave rise to a graph such that

$$\frac{n^2}{4} \approx 75000$$

The graph had about 5900 edges and thus h has a value of approximately -69000 and the graph contains about -69250 complete graphs of order three, if it contains any at all. As most of the graphs used for registering course conflicts will have the same density of edges this approach is impracticable. J. W. Moon (25) extended the work of Erdös to obtain a lower bound for the number of complete graphs of order k contained in any given graph. This, although of some theoretical interest, is useless for a starting point on any actual computation.

An alternate approach would be to attempt to determine the upper and lower limits of the chromatic number of the graph and thus, at least, find the possible range of the orders of the complete graphs. This has been made possible by the work of Ersov and Kozuhin (12) who made the following observations.

If a graph has n vertices and p edges (no loops or parallel edges) then the largest possible chromatic number, X, is:

$$X = \left[\frac{3 + \sqrt{9 + 8(p-n)}}{2}\right]$$

and the smallest possible chromatic number, x, is:

$$x = -\left[-\frac{n}{\left[\frac{n^2-2p}{n}\right]} \left(1 - \frac{\left\{\frac{n^2-2p}{n}\right\}}{1+\left[\frac{n^2-2p}{n}\right]}\right) \right]$$

where the brackets $[\ ]$ and $\{\ \}$ denote the integral and fractional parts of the number respectively. When these formulas are applied to the aforementioned data set they give the following results

$$X = 100 \qquad x = 1$$

so that

$$1 \leq \gamma(G) \leq 100.$$

These, although better than the previous bounds, are still of no great use.

Fortunately Theorem 1.5.3 makes it possible to design an algorithm to check for the existence of a complete subgraph of a particular order. To determine the largest complete subgraph it should only be necessary to make this algorithm iterative, ie. check for a complete subgraph of order n by eliminating any edges not members of at least n-2 edge circuits of length three, then iteratively entering this algorithm to check whether this reduced graph contains a complete graph of order n+1. This process is continued until the graph is composed entirely of isolated vertices, thus indicating that the largest complete subgraph is of order n-1. If a graph, G, is subjected to this operation (denoted by $\triangle$Tn) in an attempt to locate a complete subgraph of order n, then the reduced graph (in which each edge is a member of at least n-2 edge circuits of length three) will be denoted by Tn(G), a notation due to A. R. Meetham.

If the graph is stored in the form of a boolean matrix the procedure for checking that each edge is a member of at least n-2 edge circuits of length three becomes quite simple:

1/ensure that all elements of the leading diagonal of the boolean matrix, A, have the value false (this procedure is not valid for graphs with loops)

2/if an edge, α, exists between vertex i and vertex j then form a boolean vector B with n elements whose values are determined by the boolean expression

$$B_k = A_{ik} \wedge A_{jk} \qquad (k = 1,2,3,\ldots,n)$$

3/if P is the number of true elements in the boolean vector B then the edge α is a member of P edge circuits of length three.

This elementary complete graph algorithm suffers from two distinct disadvantages. The first, a rather minor disadvantage, is the fact that it will not determine a distinct complete graph. This arises from the fact that complete graphs may be interlinked. For example FIGURE 1.5.4 consists of three interlinked complete graphs of order four. The second, and major disadvantage is that the converse of Theorem 1.5.3, ie.

> If, in a graph G, all edges are part of n-2 edge circuits of length 3 then G is a complete graph of order n

is false. Any graphs whose edges are members of n-2 edge circuits of length three but do not contain a complete graph of order n will be known as "false" complete graphs. As can be seen from FIGURE 1.5.5 (the simplest known "false" complete graph) each edge is a member of two edge circuits of length three, which by Theorem 1.5.3 would indicate that it contained a complete graph of order four when, in fact, it only contains complete graphs of order three.

It is now obvious that an extra test must be incorporated in the algorithm to distingush the true from the false complete graphs. A great deal of effort was put into devising a suitable test to determine the "completeness" of the subgraphs under consideration. The possibility of simply checking all the combinations of edges and vertices was dismissed when it was found that, in a modest graph, a check would have to be done on all possible combinations of 58 vertices taken 27 at a time. This very time consuming process would have taken far longer than the original determination

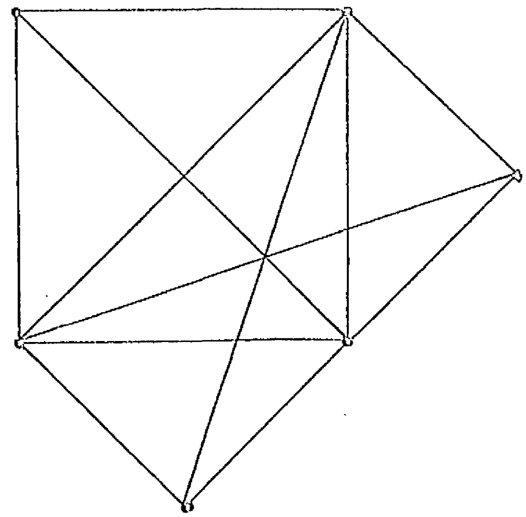


FIGURE 1.5.4

Showing three interlocked complete graphs of order four.

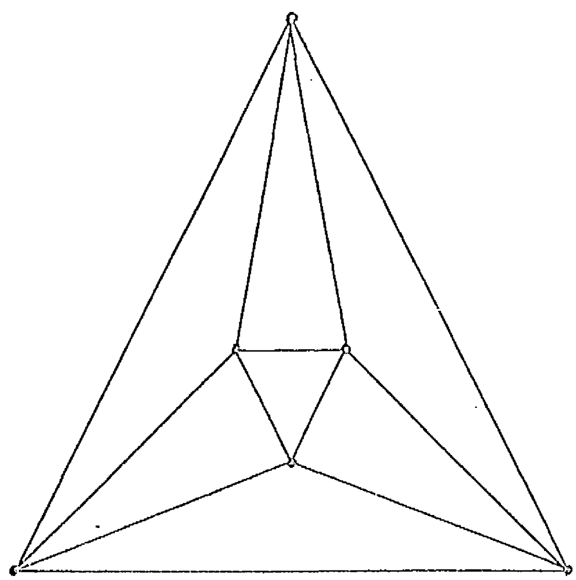


FIGURE 1.5.5

Showing the simplest known "false" complete graph.

The graphs in FIGURE 1.5.4 and FIGURE 1.5.5 have the same number of vertices and edges.

of $T_{27}(G)$.

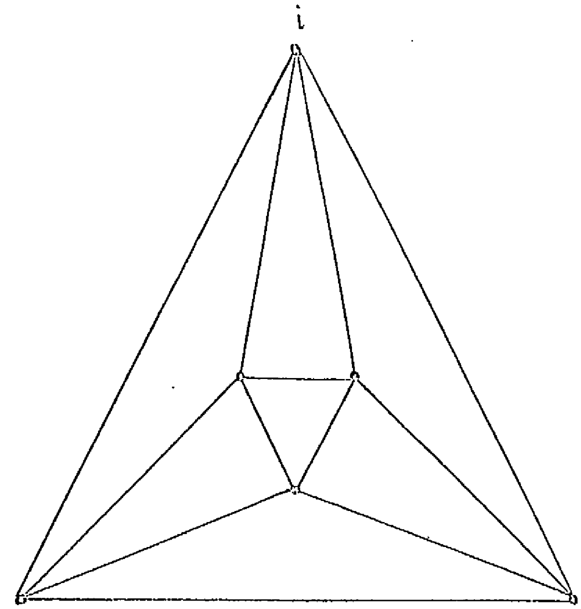
The work of Erdös, Moon and Moser, cited previously, also fails as a test for complete graphs. Both Moon's and Erdös' theorems use relations between the number of vertices and the number of edges in a graph; as the graphs in FIGURE 1.5.4 and FIGURE 1.5.5 both contain six vertices and twelve edges, even these simple examples are enough to show that the theorems by Erdős and Moon are not suitable to practical application.

The method eventually used to provide the final check for the complete subgraphs was as follows:

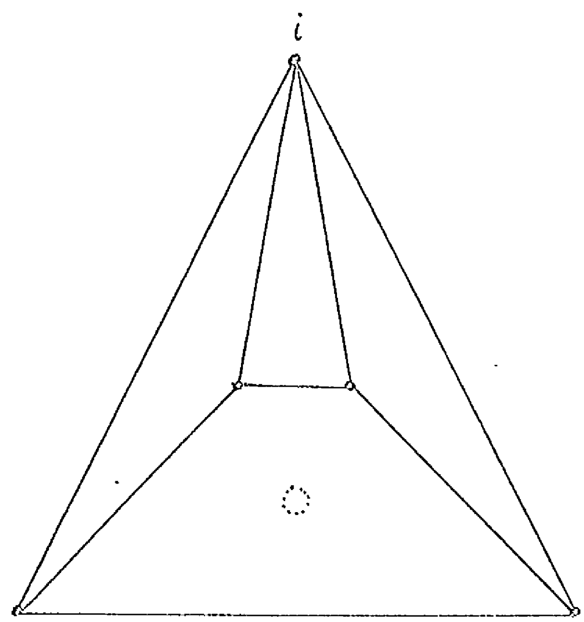
1/select a vertex, i, from $T_n(G)$ such that d_i is a minimum for all ~~i~~ the vertices in $T_n(G)$.

2/produce a graph $T_n^i(G)$, containing i and those vertices joined to i along with all the edges joining ~~the selected~~ those vertices. FIGURE 1.5.6 shows this process in graphical form.

3/a check is made to determine the number of completely connected vertices in $T_n^i(G)$. If $d_i = n-1$ then this check simply reduces to verifying that all off-diagonal elements of the boolean matrix associated with $T_n^i(G)$ have the value <u>true</u>. If $d_i \neq n-1$ then the complete graph algorithm must be applied to $T_n^i(G)$ - this amounts to a recursive entry into the complete graph algorithm and then step four is only entered when the bottom



$T_n(G)$



$T_n^i(G)$

FIGURE 1.5.6

Showing the relationship between $T_n(G)$ and $T_n^i(G)$

level of recursion has been reached.

4/if the number of completely connected vertices in $T_n^i(G)$ is greater then or equal to n then there exists a complete graph of order n. If this number is less than n then vertex i is deleted from the graph $T_n(G)$ (along with any incident edges) and the operation $\triangle T_n$ is reapplied to the now modified $T_n(G)$.

The flowcharts and actual ALGOL 60 coding are included in the appendices.

Section 1.6 Results of the Investigation

The investigation of the examination time-table problem started with data produced by means of a random number generator. This was initially set to produce three different sets of data, each of which was in the form of a boolean matrix. A pseudo-random number generator with a square distribution between zero and one provided the criterion of whether or not an edge was present in the graph, the i,j th and j,i th elements of the matrix were set to true if the random number was less than ½, 1/3, or ¼ respectively for the three different data sets.

It was soon realized that a random number generator could not simulate the cluster patterns of course conflicts that arise in a real situation, so the investigation was finally carried out on two sets of student data from the University of Alberta, Calgary (Canada) (1964 -65 and 1965 - 66 student bodies).

These two sets of data were a very good test of the procedures because the students from U.A.C. are able to attend classes from many different disciplines and thus the graphs of course conflicts are much more complex than those arising out of universities with a rigid faculty structure. The procedure for separating a graph into its individual connected subgraphs, as described in Section 1.1, was implemented and both data sets subjected to this separation process. It was found that, for all practical purposes, the two graphs from U.A.C. were nonseparable.

No attempt was made to reduce these two graphs by finding minimal articulated sets or cut sets because the procedures seemed to function well even on these large singly connected graphs. However if the size of the graph grew by a factor of two or more it would be necessary to attempt some form of reduction simply because most computer memories could not hold it all at once. This could raise serious problems for a very large connected graph as most of the reduction procedures are based on the matrix method of storing the data, thus reduction presents the same fundamental storage problem as the original colouring procedures.

An examination time-table was produced for each of the ~~three~~ two sets of data, using first the Peck-Williams procedure and then the eigenvector approximation procedure. It was found that, in both cases, the eigenvector approximation procedure produced an examination time-table using less periods than the time-table produced by the Peck-Williams procedure (see TABLE 1.6.1). The data was then subjected to the complete graph procedure, in order to determine a lower bound for the number of examination time periods required for each data set.

	Peck-Williams	eigenvector approximation	largest complete graph
U.A.C. 64 - 65 (547 vertices)	27	26	22
U.A.C. 65 - 66 (656 vertices)	29	28	25

TABLE 1.6.1

Showing the number of examination time periods required by the different procedures on each of the data sets, along with the size of the largest complete graph in each set.

	Peck-Williams	eigenvector approximation
U.A.C. 64 - 65	5 min. 8 sec.	5 min. 42 sec.
U.A.C. 65 - 66	6 min. 6 sec.	6 min. 49 sec.

TABLE 1.6.2

Showing the run times of the different procedures on each data set.

From TABLE 1.6.1 it can be seen that the eigenvector approximation procedure came closer to the theoretical minimum than the Peck-Williams procedure. The discrepancy between the actual and theoretical results can be conjectured to be due to the fact that the critical chromatic subgraph of the data sets is not a complete graph but rather a complex graph containing a complete graph as a subgraph. If this conjecture is correct, and there seems no way of testing its truth, then the lower bound is raised slightly and better agreement would be obtained between actual results and the absolute theoretical minimum.

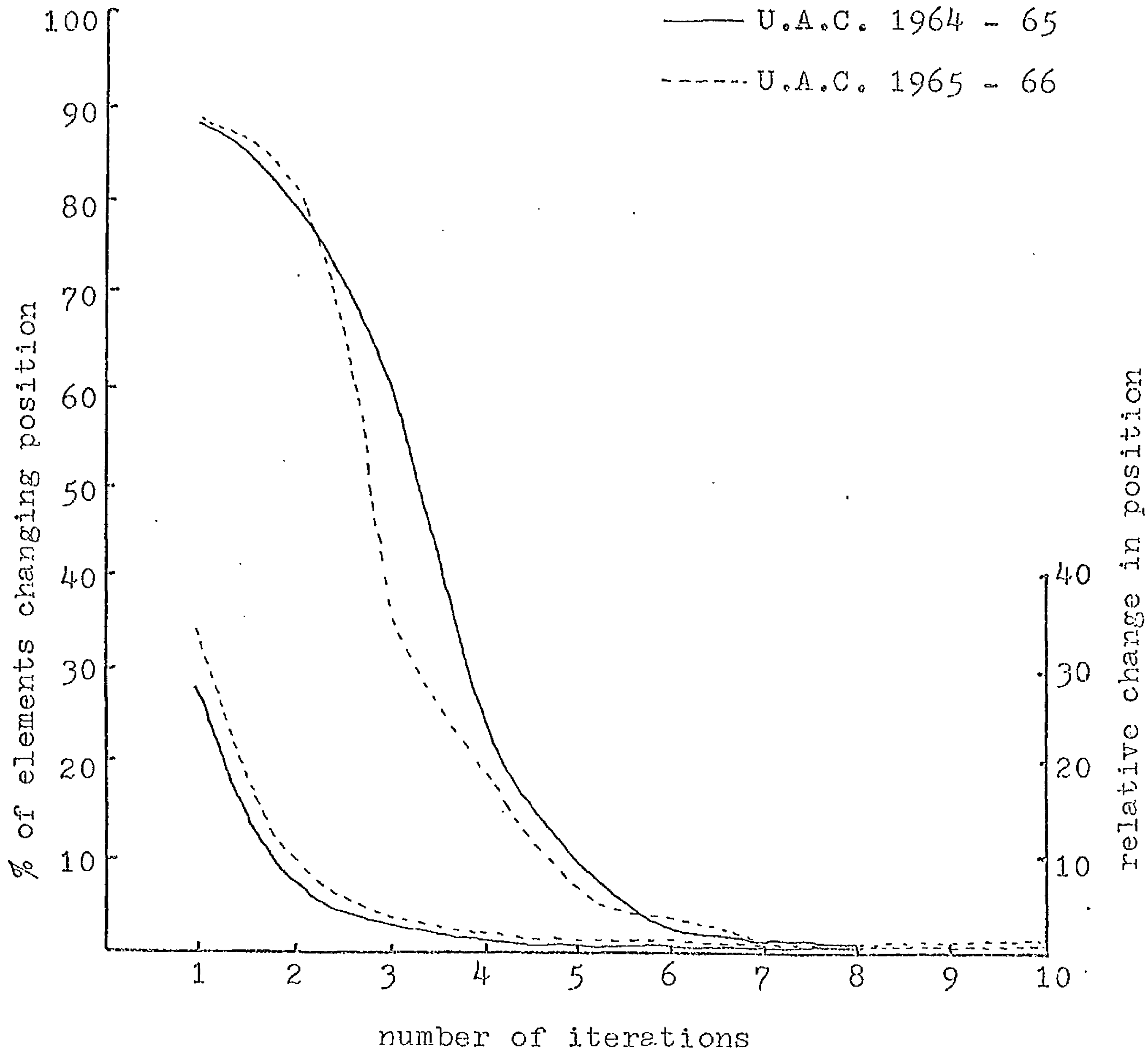
A very intensive study was made on the U.A.C. 1965 - 66 data set to try to discover if the chromatic number was indeed larger than 25 (the size of its largest complete subgraph). The study revealed that the largest complete subgraph was of order 25 but there existed three subgraphs of order 26 lacking only one edge each to make them complete graphs, two subgraphs of order 27 lacking only two edges each to make them complete, and one subgraph of order 28 lacking only three edges to make it complete. Of all the complete subgraphs of order 25, eight were found, all completely interlocking. This very complex situation is exactly what is required as a base for a complex critical chromatic subgraph of order greater than 25. A great deal of work was put into an attempt to elucidate the structure of this critical chromatic subgraph but it could not be found. This failure to find a critical k-chromatic subgraph ($k > 25$) does not mean that it did not exist, the complex structure of the subgraphs of order 25, 26, 27, and 28 point to its existence but the number of possible subtle combinations of these subgraphs with any of the 600 other vertices makes its determination hinge on having very extraordinary luck.

TABLE 1.6.2 indicates the running time for each of the examination time-table procedures on each of the data sets. This is the time taken on an English-Electric-Leo-Marconi KDF 9 computer with the programs written in the KIDSGROVE (unoptimised) dialect of ALGOL 60. The very large boolean matrices (656 X 656 in the case of the U.A.C. 1965 - 66 data) were kept in the store by designing a series of ALGOL procedures written in USER CODE (the assembly language of the KDF 9) which packed a single element of the matrix into one bit. Thus the 430,336 elements of the matrix could be contained in approximatly 10,000 KDF 9 48-bit words. This packing of data is, unfortunatly, necessary because the procedures must examine the matrix elements at random and, if the matrix were stored on magnetic tape or even a random access device, the time taken to produce a time-table would make the procedures uneconomic, unless implemented on a time-shared machine.

The actual calculation of the next approximation to the largest eigenvector of the system was accomplished by a highly efficient procedure written in USER CODE. This procedure made special use of the fact that the matrix associated with the graph was stored one element per bit, and this, combined with the powerful set of logical instructions on the KDF 9, has resulted in an ultra fast routine. Thus to implement the eigenvector approximation procedure on an alternative computer may increase the computation time over the Peck-Williams procedure by a greater percentage than is evident from TABLE 1.5.2.

The effect of continued iteration towards the eigenvector corresponding to the largest eigenvalue of the associated matrix was examined in some detail for the two data sets. In particular the relative magnitudes of the elements of this vector were investigated at each iteration, because of their importance in controlling the order in which the vertices are chosen for assignment. If the relative magnitudes of the elements of this vector remain the same after one or two iterations then it is senseless to continue iterating towards the actual eigenvector when even a very poor approximation is computationally satisfactory. A program was written to compare the relative magnitudes of the elements from one iteration to the next and the results produced are shown in FIGURE 1.6.1. The percentage of elements changing position of relative magnitude and the average number of places changed in the scale of relative magnitudes shed some light on the computational effort needed to produce the gains offered by the eigenvector approximation colouring procedure. It is evident that the ordering goes through a dramatic resequencing during the first few iterations. The first iteration changes the order of about 85% of all the vertices by an average of approximately 30 places up or down the list. However by the time five iterations have been done only about 10% of the vertices are changing their positions in the table of relative magnitudes and this change is not more than one or two places. The actual eigenvector is found (to an accuracy determined by the KDF 9 48-bit word length) after 10 - 15 iterations, thus it is only necessary to carry the iteration out a few times to reap the benefit of any gains of the more advanced colouring procedure.

The 1964 - 65 data was used by the University of Alberta, Calgary to produce an examination time-table by their traditional methods. This was done by two highly competent (and as a result highly paid) members of the Office of the Registrar in slightly less than six weeks. The resulting time-table had 30 examination time periods, and did not satisfy the no-conflict demands of about 15 students. When this is compared to the cost and efficiency of the computer procedure it can easily be seen that the computer can save a



U.A.C. 1964 - 65
U.A.C. 1965 - 66
100
90
80
70
60
50
40
30
20
10
% of elements changing position
40
30
20
10
relative change in position
1
2
3
4
5
6
7
8
9
10
number of iterations

FIGURE 1.6.1

large amount of money, time, and effort in this area of a university's administration. This last statement does not take into account the costs of data preparation, however the data may be easily obtained as a byproduct of the sectioning process described in Chapter 2.

During this investigation it was noted that every examination time-table procedure mentioned in the literatur had, as one of its parameters, an indication of the number of seats available in the examination room. For the basic "one class - one paper" situation there is absolutely no need to consider the room size as a parameter. If the number of students scheduled to write examinations in a particular period exceeds the capacity of the rooms available, then the registrar may reschedule some of the examinations from this period to another free day and still be assured that, in his situation, he has a near optimum time-table. It was found that, for the U.A.C. data, the actual physical situation contained so many conflicts that using a cut off parameter such as room size was unnecessary. In fact even using the size of the largest class as the room size parameter, the time-table produced did not vary in the number of periods used.

Section 1.7 Possible Modifications

The system for finding examination time-tables, as described in the previous sections, seems to be better (in regard to computing power necessary and number of periods used) than any described so far in the open literature. However a number of practical objections may be made to it. It is often necessary to limit the number of examination time periods, or attempt to distribute the number of examinations evenly throughout the examination session, or cause two different examination to be held at the same time.

The last of these objections may easily be dealt with by some form of pre-assignment feature as in the published version of the Peck-Williams procedure. The other two objectives, limiting the number of examination time periods and distribution of the examinations, are by their very nature contrary to the fundamental idea behind the colouring procedure.

The number of time periods, or colours, may not be limited because the procedure will already attempt to use the minimum number of periods. An attempt may be made to determine the bounds of exactly how many periods will be used but these bounds are very poor and their determination takes more effort than the actual computation of the time-table. The even distribution of the examinations is also a point which cou[illegible] not

M.R. WILLIAMS

Author: * Also Negative Microfilm at end of run.

Title: A GRAPH THEORY MODEL FOR THE COMP. SOLUTION...

Date of deposit: 12.2.69

If you remove this volume from the shelf, enter on this card the reason for removal, the date, and your initials and leave the card on the shelf in place of the volume. Before re-shelving the volume, score through the record of removal and replace the card in the volume.

Reason for removal	Date	Initials
~~R.C. Ref. Desk~~	~~3.11.75~~	~~MMcD.~~
~~[illegible]~~	~~7-7-75~~	~~LB.~~
~~Ref Desk~~	~~10.4.76.~~	~~SR~~
~~ILL - Ankara~~	~~31.8.76~~	~~ZB~~
~~[illegible]~~	~~[illegible]~~	~~[illegible]~~
~~[illegible] - Presque Isle Maine~~	~~10/3/78~~	~~LM~~
~~RC: Photocopying~~	~~4/5/78~~	~~LM~~
~~RC~~	~~3/9/82~~	~~[illegible]~~
~~Photocopying~~	~~23/7/82~~	~~L.H~~

Process	Date	Initials
Accessioned		
Catalogued		
Classified		
Cataloguing checked		
Processed		
Processing checked		
Shelved		

be changed without disturbing the optimising effect of the procedure. After the time-table has been found the possibility exists of moving some of the examinations in the early periods to the later periods without causing conflicts. This may help to ease the load on the first few periods.

FIGURE 1.7.1 shows the distribution of examinations throughout the time-table. This top heavy form is appreciated by most of the staff but definitely not welcomed by the students.

If the university administration is willing to allow a few conflicts in their examination time-table, both of these objections can be alleviated, but only at a price. As can be seen from FIGURE 1.7.1 the last few time periods only contain a few examinations. It would be possible to take the time period with the least number of examinations and place these in other periods in such a manner that the number of conflicts generated is a minimum. This redistribution will require a matrix, C, such that c_{ij} is the number of students involved in the conflict between course i course j, so that the number of students involved in any one conflict may be noted. The size of this matrix dictates that it will have to be kept on some form of auxiliary memory, probably a random access device. However some packing of the elements is possible because the largest element is known to be

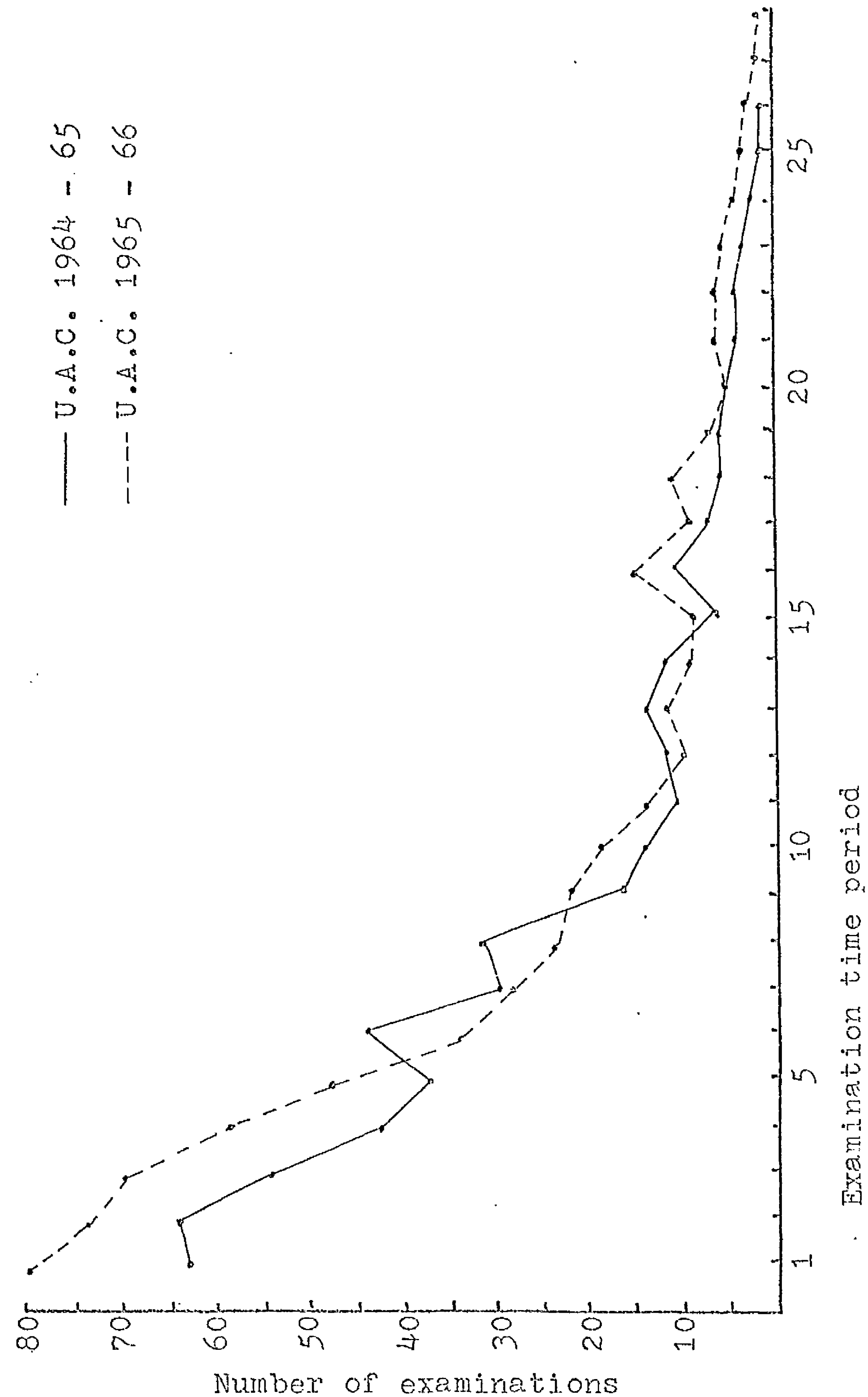


FIGURE 1.7.1

Showing the number of examinations in each time period less than or equal to the enrolment in the second most popular course. Another requirement of the redistribution is that all examinations being removed from the Ith period must be placed in periods that have not had other examinations from the Jth period already put there. Failure to observe this restriction may lead to three and four way conflicts instead of just the two way conflicts being formed. To reduce the number of time periods by the greatest amount for the least cost, all the examinations from period I should be redistributed to the same new period.

CHAPTER 2

Sectioning Students to Classes

Section 2.1 The Problem

Many universities overcome their time-tabling or space problems by splitting each class into many different sections, each given at a different time throughout the week. For example, if a first year course in chemistry is expecting an enrolment of 200 students, but the lecture rooms will only hold 75 people and the laboratories will only accommodate 50 students at a time, the university administration may section the class as follows:

Lect. section 1 - Mon. Wed. Fri. Room 813 9:00-10:0

Lect. section 2 - Mon. Wed. Fri. Room 93 11:00-12:0

Lect. section 3 - Tues. Thurs. Sat. Room 798 10:00-11:0

Lab. section 1 - Mon. Room 712 9:00-12:00

Lab. section 2 - Tues. Room 712 9:00-12:00

Lab. section 3 - Wed. Room 712 2:00-5:00

Lab. section 4 - Fri. Room 712 2:00-5:00

If a student was taking mathematics on Monday morning at 9:00 then he could still easily fit one of the other two sections of chemistry lecture into his time-table. It is now necessary to find some method of distributing the students into the various sections which will not only take into account the 12 possible choices of chemistry but the total number of choices available to him from all his subjects.

Unfortunatly this choice of sections can not be left to the student himself. Few students would willingly choose chemistry lecture section 3, and thus there would be hopless overcrowding in lecture sections 1 and 2.

Sectioning students to classes with the aid of a computer has been dealt with in many papers and actually implemented on a few machines. The majority of these implementations and discussions have dealt with the problem in the situation where large scale computers were available. In this discussion a suggestion will be given involving a large scale computer and a suggestion which will enable its implementation in the situation in which a small scale computer with a random access auxiliary memory, or a large scale computer (on which time is a very important factor) is available to the administration.

All the methods under consideration require the university administration to supply a master time-table, giving the times and maximum enrolments for each section of each course, and a set of cards (or other suitable input medium) for each student indicating the courses in which he is enrolled.

The major objective of computer scheduling is to assign the student to nonconflicting sections of his courses, subject to some or all of the following conditions:

1/When the sectioning of all students has been finished the different sections of a class must have roughly equal numbers of students, or the numbers of students in the different sections must be in a predetermined ratio to one another.

2/Particular students should be placed in a particular section of a course, the constraint being sex, faculty or other personal or academic information.

3/Particular sections of a course are closed to particular students, the constraint again being sex, faculty, or other personal or academic information.

One of the benefits of sectioning using a computer is that the registration process may be speeded up, but this is not the only advantage. If, as each student is sectioned, a record is made of his section assignments, the procedure may have the beneficial side-effect of providing accurate student records, statistics, and class lists immediately upon the conclusion of the registration procedure.

As in the problem of examination time-tables, the master time-table of the university classes may be best described in the terminology of graph theory. The master time-table may be visualized as a graph, $G = (V,U)$, whose vertices, V, are the classes offered by the institution, and the edge generating function, U, being defined as generating an edge between v_i and v_j if both course i and course j are offered during the same or overlapping time periods.

Sectioning students to classes now becomes the problem of partitioning the graph of the master time-table into disjoint sets such that no two vertices in any given set are joined by an edge. It can now be seen that this is exactly the same colouring problem as was presented in the discussion on examination time-tables. However, because of the different physical situation in which the problem arises, and the added constraints to its solution, the actual procedures used for the solution will differ from those used in the solution of examination time-tables.

The more complicated system of courses with multiple sections is a simple extension of the aforementioned graph, namely each of the sections of a course is now a separate vertex. The vertex corresponding to section k of course i will be denoted by v_i^k.

Section 2.2 Heuristic Procedures

The first, and perhaps most general, attack on the problem was initiated in 1959 at Purdue University by J. F. Blakesley (3), who although handicapped by an extremely primitive computer, produced a system which embodied the basic design of every subsequent heuristic implementation. The program that he developed follows the logic

1/each student must be sectioned and the procedure used should be as fast as possible

2/the last student sectioned should have the same probability of being assigned to a particular course section as the first student had.

From this logic two key points emerge. One is that courses must be ordered according to the difficulty of finding alternate sections (single section courses first, for they have no alternate time schedule, followed by courses with more and more sections). The second, and just as important, point requires that the student be placed in the section with the largest number of remaining unfilled student places. If this section cannot be made to fit, then the remaining sections are tried (from largest to smallest number of remaining places) until a section is found which will fit the schedule.

FIGURE 2.2.1 is a simplified flow chart of the basic Blakesley model for a computer sectioning program. The Blakesley model consists of three major loops, which try all possible combinations of courses in an attempt to construct a schedule. These loops are:

A/the primary course section assignment loop

- select the section (of the course under consideration) with the largest number of unfilled student places
- if the selected section does not conflict with the previously assigned course sections then proceed on to the next course, otherwise enter loop B

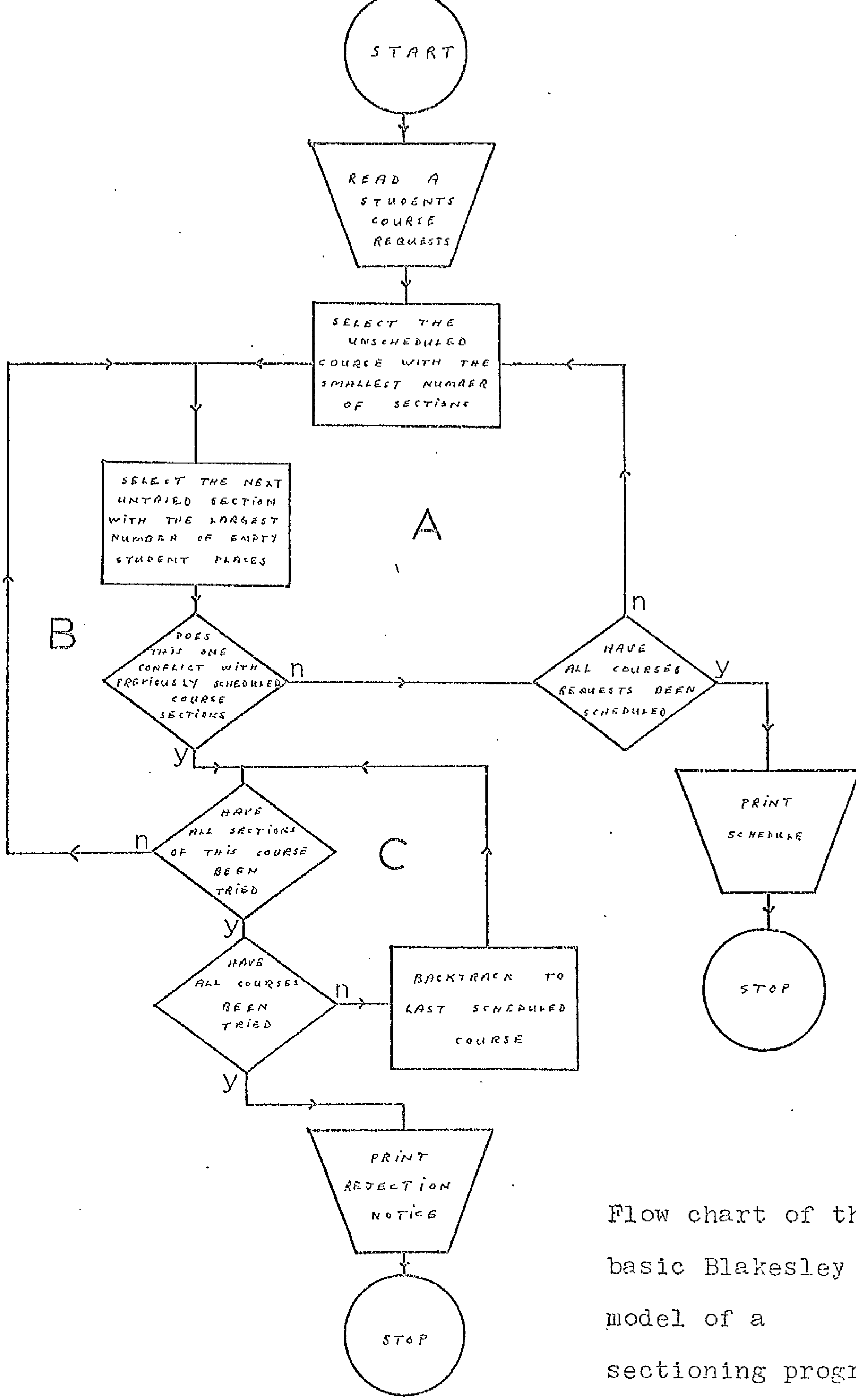
B/the section progression loop

- this selects the section with the next largest number of empty student places and returns to loop A

C/the course backtrack loop

- this loop is used when all sections of a course have been tried and found to conflict with the sections of previously scheduled courses - the loop backtracks to the last scheduled course, selects an alternate non-conflicting section and returns to loop A to schedule the subsequent courses.

The basic plan of the Blakesley sectioning program, usually termed a yo-yo tree search, is common to all



Flow chart of the basic Blakesley model of a sectioning program.

FIGURE 2.2.1

environments in which the sectioning procedures may be implemented, that is to say, it is not dependent on the detailed faculty or course structure of an institution. Every proposed sectioning procedure has followed the same general outline, the only difference being a gradual refinement and modification of the methods of accomplishing the three basic steps.

The two basic heuristics (ordering the student course requests by the number of sections in each, and selecting for consideration the untried section with the smallest enrolment) have also been used in all the procedures investigated, the sole exception being a very early experiment at Washington State University (13).

It is rather curious that Blakesley used the number of empty seats left in a class, rather than the number of people assigned to that class, as his criterion for choosing the section with the smallest enrolment. The author's own work (hereafter refered to as the Colijn-Williams procedure) points out the pit-falls of the use of room capacity. A quote from the paper by Colijn and Williams (6) will show the havoc that the use of room capacity can cause.

> "The University Administration had produced a master time-table showing a particular course with three sections, each of which was to have an enrolment of about thirty students. Sections 1 and 2 were to meet in rooms

holding forty students each but, due to limitations of the physical plant available, section 3 had to meet in a room accommodating 250 people. The result of sectioning using room capacities, as the section sorting criterion, was that approximately 90% of the students requesting this course were assigned to section 3, leaving sections 1 and 2 with only those students whose other course requests blocked section 3 from consideration."

Another curious point about Blakesley's original system is that it would consider all possible combinations of the available course sections before it decided that a time-table could not be produced. If a student is enrolled in a series of courses, all of which have a large number of sections, then it is possible to have in excess of 10^9 section combinations to be investigated. Even very large computers can only be expected to consider a fraction of these. It was not until quite recently that Faulkner (13) and Colijn and Williams (6) attempted to design the backtracking loops to search intelligently for a workable schedule rather than trying an arbitrary number (5000 in the case of Anderson (1)) of combinations before giving up.

Since Blakesley's first attempt the method of storing the master time-table information has undergone a vast improvement. Although the storage of the master time-table is a practical matter, depending heavily on the computer to be used, it is of sufficient significance to be briefly discussed. The significance lies in the fact that it requires a large amount of memory, and that it must be accessed frequently. Because of these heavy expenditures of the computer's resources on the time-table, the mode of storage will affect the efficiency as well as the basic design of the final procedure to a marked degree.

Several methods are available for storing the time-table information. The first, that of recording the actual day and time of a course, eg. M.W.F. 9-10 (or suitable coding for this information, as was done by Blakesley) will not be considered because of the obviously greater ease of processing offered by the other methods.

The second method involves the use of a boolean "time-vector", T, of which two types have been used in the past.

> Type 1 - 26 boolean elements, six of which represent the days of the week (ie. T_i = <u>true</u> if the class is held on the i th day of the week, $1 \leq i \leq 6$) and 20 representing the time of day. The first of these time of day elements represents the half hour period 8:00 - 8:30, the second 8:30 - 9:00, the last element representing the period 5:30 -

6:00. For example, a class given Monday, Wednesday, and Friday from 9:00 - 10:00 would have a type 1 time-vector of:

101010001100000000000000000

Type 2 - Similar to the "time of day" elements in the type 1 time-vector, except there is a complete set of 20 booleans for each day of the week. Thus the type 2 time-vector consists of 120 boolean elements, the first 20 representin the time of day the class is given on Monday, the next 20 representing Tuesday, the final 20 representing Saturday. For the example given under type 1, the type 2 time-vector would be:

0011000000000000000000000000000000000000110
......et

It is easily seen that, although type 2 time-vectors take more memory space than type 1, they allow much more flexibility in representing a class which is given on different times on different days. If the vertices of the time-table graph each have an associated time-vector, then by doing a boolean AND operation on time-vector i and time-vector j (in the case of type 1 time-vectors this must be done twice, once on the day elements and once on the time elements) it is possible, by checking to see if the resultant vector has any true elements, to determine if an edge exists between vertex i and vertex j of the time-table graph.

A third method of storing the time-table involves the use of the boolean matrix associated with the graph. The use of a boolean matrix for storing time-table information, although more difficult to set up than the time-vector methods, has several advantages which will be explored in a later section. It also has the disadvantage that unless the university offers fewer than 120 courses it takes considerably more storage than the time-vector system. This, however, need not be a serious drawback if a high speed random access device is available.

A second, and perhaps better known, attack on the problem was initiated by Anderson (1) in conjunction with the New England School Development Council in 1962. Anderson's procedure was extremly fast - up to 1000 pupils per minute - but it should be remembered that this speed was due, to a large extent, to the fact that it was a school sectioning problem rather than a university one; the master time-table for a school will invariably be better suited to machine sectioning because of the fewer courses offered, each course generally having only one section, and the more limited choice given to the pupils.

As well as the usual master time-table information (eventually coded into a type 2 time-vector) Anderson also recorded two other variables, X_i^k and Y_i^k, where X_i^k is the number of pupils that must be registered as wishing to take section k of course i before it becomes a practical proposition to even offer this section, and Y_i^k is the maximum number of pupils that may be admitted to section k of course i. If the first T-1 courses, for a particular student, have been successfully sectioned, then Anderson's procedure will perform the following operations in attempting to determine a valid section assignment to course T.

1/ Select, for consideration, all possible sections of course T such that

$$X_T^k > 0 \quad (k = 1,2,3,\ldots,n)$$

(this is known as the minimum mode of search).

2/ Order the course sections, obtained in the previous step, in numerically descending order by their respective X_T^k.

3/ Compare the time-vector of the first section (the one with maximum X_T^k) with the time-vectors of the sections scheduled for the previous T-1 courses to determine if it conflicts with any of the previously assigned sections. If this section will not "fit", repeat the procedure for all the permissible sections of course T. If the list of permissible sections is exhausted without finding a fit go to step 4. If a fit is found decrease X_T^k and Y_T^k by one and attempt to schedule course T+1.

4/ Redefine the list of permissible sections as all sections such that

$$Y_T^k > 0 \quad (k = 1,2,3,\ldots,n)$$

(this is known as the maximum mode of search). Repeat steps 1 to 4, if this does not produce a fit then it is assumed that it was the assignment to course T-1 which is the cause of the conflict, therefore Anderson exchanges course T with course T-1 and tries again to find a workable set of section assignments for this student. If, after trying 5000 possible section combinations, a time-table is not found the student is abandoned and his time-table must be prepared by hand.

The values given to the individual X_i^k and Y_i^k will dramatically alter the efficiency of the system. For example if n pupils request course T (which has p possible sections) then if

$$\sum_{k=1}^{p} Y_T^k < n$$

it will result in pupils not being sectioned due to insufficient seats available; if

$$\sum_{k=1}^{r} X_T^k < n$$

it will force the procedure into the maximum mode of search when this action is not necessary. For single section courses it would make the computer look for alternative sections when there are none. On the other hand a much finer degree of control can be kept on the sectioning process with modifications being made to the individual X_i^k and Y_i^k. For example, section imbalance may be corrected thus:

- a larger X_i^k will cause section k to fill up faster;
- a smaller X_i^k will cause this section to remain empty for a longer period;
- a certain section, k, (known to fit the schedules of difficult pupils) may be kept open because the procedure will reserve $Y_i^k - X_i^k$ seats for students whose schedules are determined in the maximum mode of search.

However, if, as Anderson suggests, his procedure will deal with up to 1000 students per minute, then any thought of modifying the values of X_i^k and Y_i^k during the execution of the program is ridiculous. If it is implemented on a much slower machine than Anderson's I.B.M. 7094 then conceivably there may be enough time available for a human being to use his abilities and knowledge to modify the parameters during the execution of the program.

Several other attempts at designing class sectioning programs have been made in the interval since Anderson's report was published. Although these seem to have been quite independent attacks, it will suffice to describe in detail the procedure proposed by Colijn and Williams (6) as this contains most of the devices used by other authors plus one or two extra interesting heuristic steps.

The Colijn-Williams procedure was based on a type 2 time-vector as the method of storing the master time-table information. Their procedure was designed to be implemented on a small I.B.M. 1620 computer at the University of Alberta, Calgary. The computer configuration (40,000 digits of core store, one disk unit of 2,000,000 digits storage, 240 line per minute printer) was very limited, in particular the digital form of the core store created problems with the necessary boolean and logical operations. The I.B.M. 1620 had no arithmetic unit, instead it used a series of tables, located in the core store, to look up the answer to each arithmetic operation a digit at a time. This peculiar feature was used, by a suitable modification of the arithmetic tables, to effectively change the ADD instruction to a boolean AND instruction, and to change the COMPARE instruction into one which would set an indicator if all 120 booleans were zero.

The course structure they had to deal with complicated the sectioning procedure to such an extent that a "pure" heuristic was impossible to design. The University offered 600 courses, which were divided into about 1500 sections, however a large number of these sections were reserved for students of a particular sex or studying particular subjects. A further restriction pertained to students in the Faculty of Education who could not attend classes offered by other faculties if they were held in the mornings (because of their student teaching requirements) however they were allowed to attend morning sessions of some Education classes. To complicate matters further some classes were held only in the first term (Oct. - Jan.), some only in the second term (Feb. - May), and some all year. Because the student teaching was only held in the mornings of the second term, it further complicates the sectioning of the students in the Faculty of Education. It is instructive to examine the sectioning problem under these conditions because they are the type of constraints found in practice, and any study of the pure situation would lead to possible false conclusions.

It is also instructive to examine the sectioning problem when only a small computer is available. Due to the relatively slow access time for the 1620 disk unit the requests for time-vector information must be kept down to an absolute minimum. Thus short cuts had to be made in the heuristics which, had there been a large scale computer, would have been overlooked as being trivial.

The procedure starts by reading the student's course requests and forming a vector, REQUEST, such that the i th element of this vector contains the code number of the i th course request and the number of sections in that request. This REQUEST vector is then subjected to the standard heuristic of sorting its elements into ascending order by the number of sections in each request. This ensures that the single section courses will be dealt with before the courses that have a number of different possible sections.

The course requests are dealt with one at a time, one section of each course being chosen for the student's time-table. This sorting heuristic is effective in producing a time-table for about 75% of the cases but, for the other 25% some sort of "back up and try again" process must be attempted.

On the basis of the heuristic assumption:

> "If a set of course assignments can be made, there is at least one correct order in which to process the requests such that the schedule

to the relativly slow access time for the 1620 disk unit the requests for time-vector information must be kept down to an absolute minimum. Thus short cuts had to be made in the heuristics which, had there been a large scale computer, would have been overlooked as being trivial.

The procedure starts by reading the student's course requests and forming a vector, REQUEST, such that the i th element of this vector contains the code number of the i th course request and the number of sections in that request. This REQUEST vector is then subjected to the standard heuristic of sorting its elements into ascending order by the number of sections in each request. This ensures that the single section courses will be dealt with before the courses that have a number of different possible sections.

The course requests are dealt with one at a time, one section of each course being choosen for the student's time-table. This sorting heuristic is effective in producing a time-table for about 75% of the cases but, for the other 25% some sort of "back up and try again" process must be attempted.

On the basis of the heuristic assumption:

> "If a set of course assignments can be made, there is at least one correct order in which to process the requests such that the schedule

will be produced with the least duplication of effort."

the following backtracking heuristic was developed (the heuristic is difficult to describe but reference to the simplified flow chart, FIGURE 2.2.2, should help).

When course N can not be given a conflict-free assignment, the list of previous assignments is scanned to find a course, M, whose assignment caused the conflict. If N is a multisectioned course then it is possible, and in fact likely, that section N_1 conflicts with the assignment made for course M, while section N_2 conflicts with the assignment made for course L, thus an arbitrary section of N must be used for the backward scan.

If it is found that both course M and course N have only a single section, then it is useless to continue and the student is told to drop either M or N and select another course. If course N has multiple sections, and course M has only a single section, then it is clear that the assignment for M can not be changed and thus a different section of N must be used for the scan back. This process is continued until a previous assignment of a multiple sectioned course is found which blocks course N from assignment. If all the sections of N are blocked out by single section courses, then again it is useless to continue and the student is informed of the multiple course conflict.

After the conflict has been found the problem is

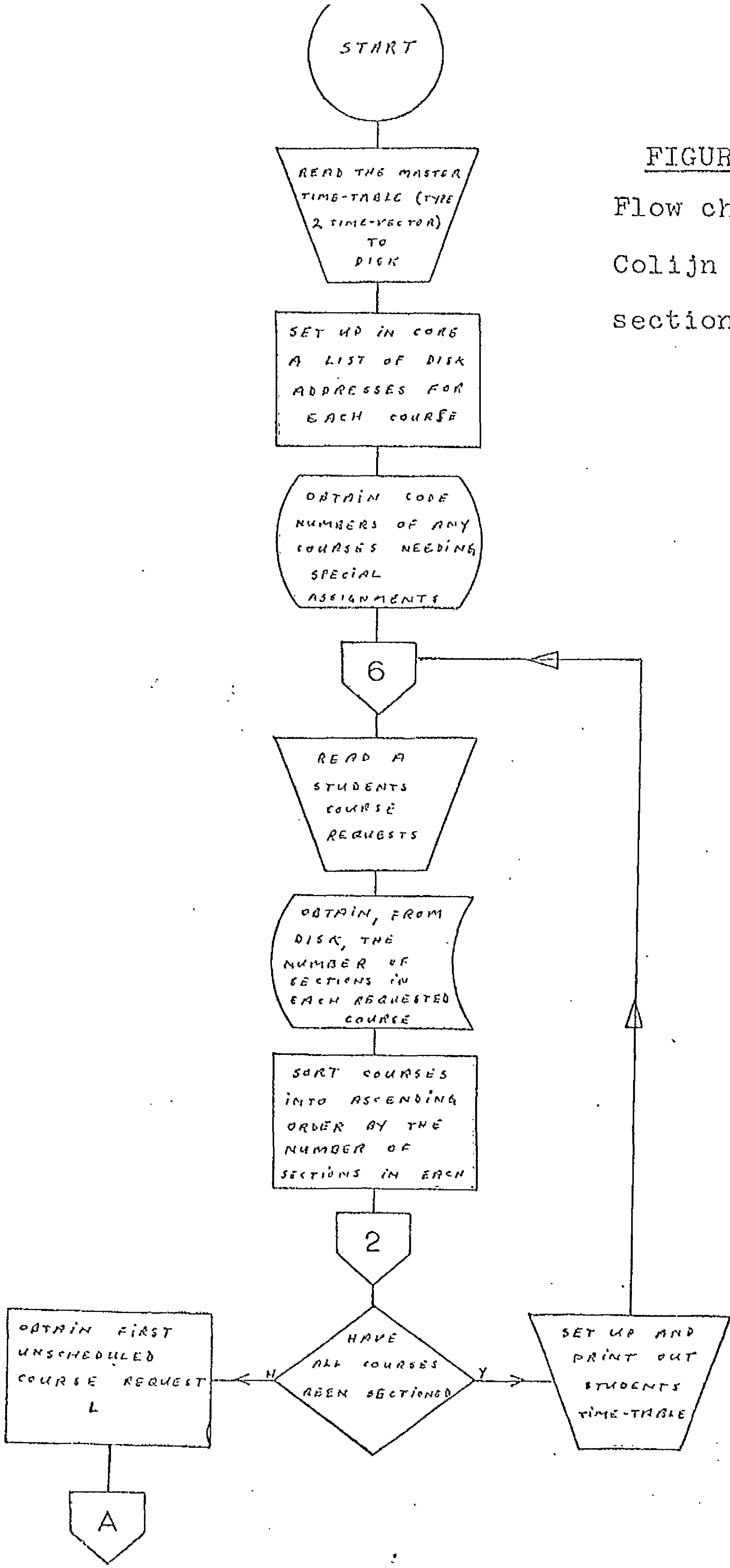


FIGURE 2.2.2 (1)

Flow chart of the Colijn - Williams sectioning procedure

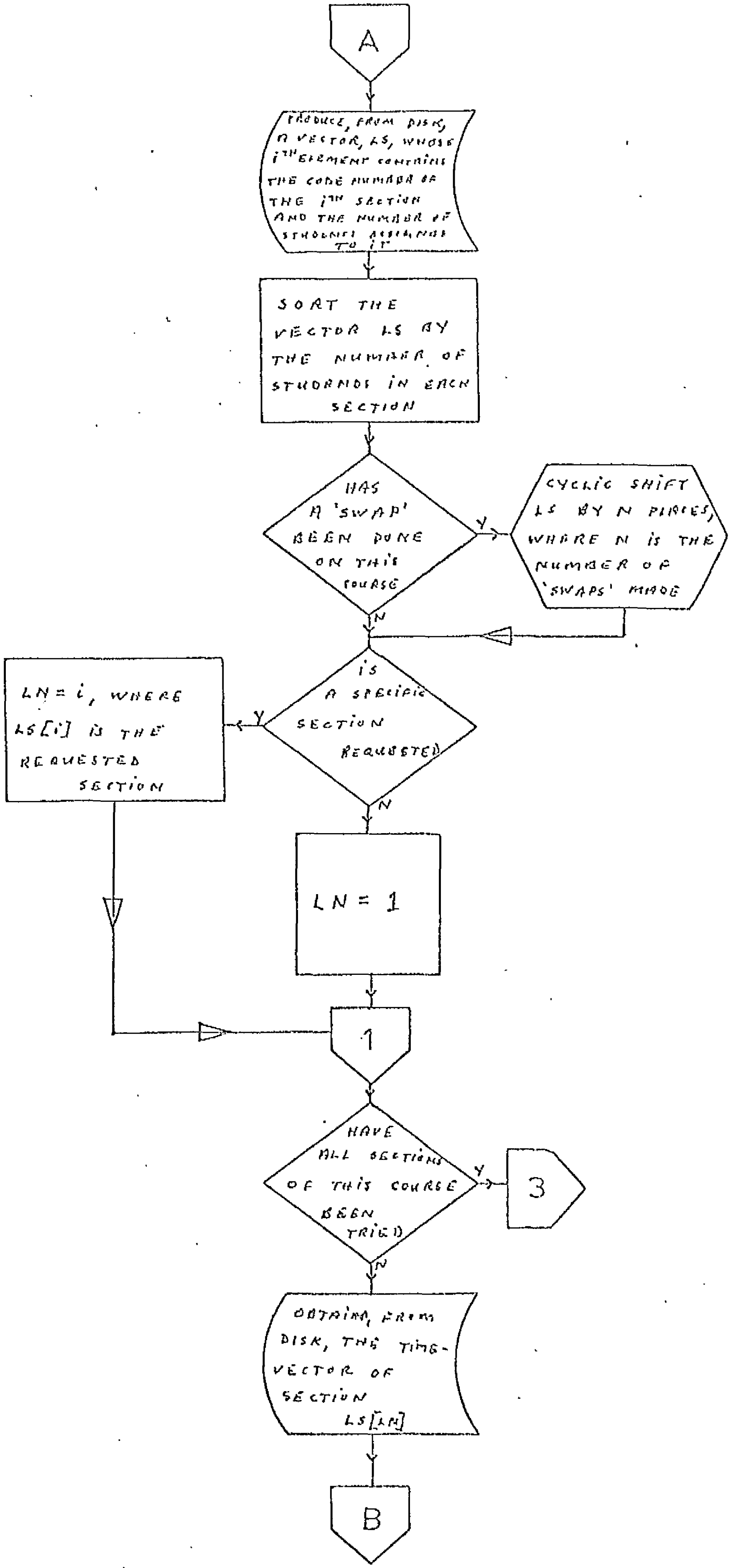

A
PRODUCE, FROM DISK, A VECTOR, LS, WHOSE iTH ELEMENT CONTAINS THE CODE NUMBER OF THE iTH SECTION AND THE NUMBER OF STUDENTS ASSIGNED TO IT
SORT THE VECTOR LS BY THE NUMBER OF STUDENTS IN EACH SECTION
HAS A 'SWAP' BEEN DONE ON THIS COURSE
Y
CYCLIC SHIFT LS BY N PLACES, WHERE N IS THE NUMBER OF 'SWAPS' MADE
N
IS A SPECIFIC SECTION REQUESTED
Y
LN = i, WHERE LS[i] IS THE REQUESTED SECTION
N
LN = 1
1
HAVE ALL SECTIONS OF THIS COURSE BEEN TRIED
Y
3
N
OBTAIN, FROM DISK, THE TIME-VECTOR OF SECTION LS[LN]
B

FIGURE 2.2.2 (2)

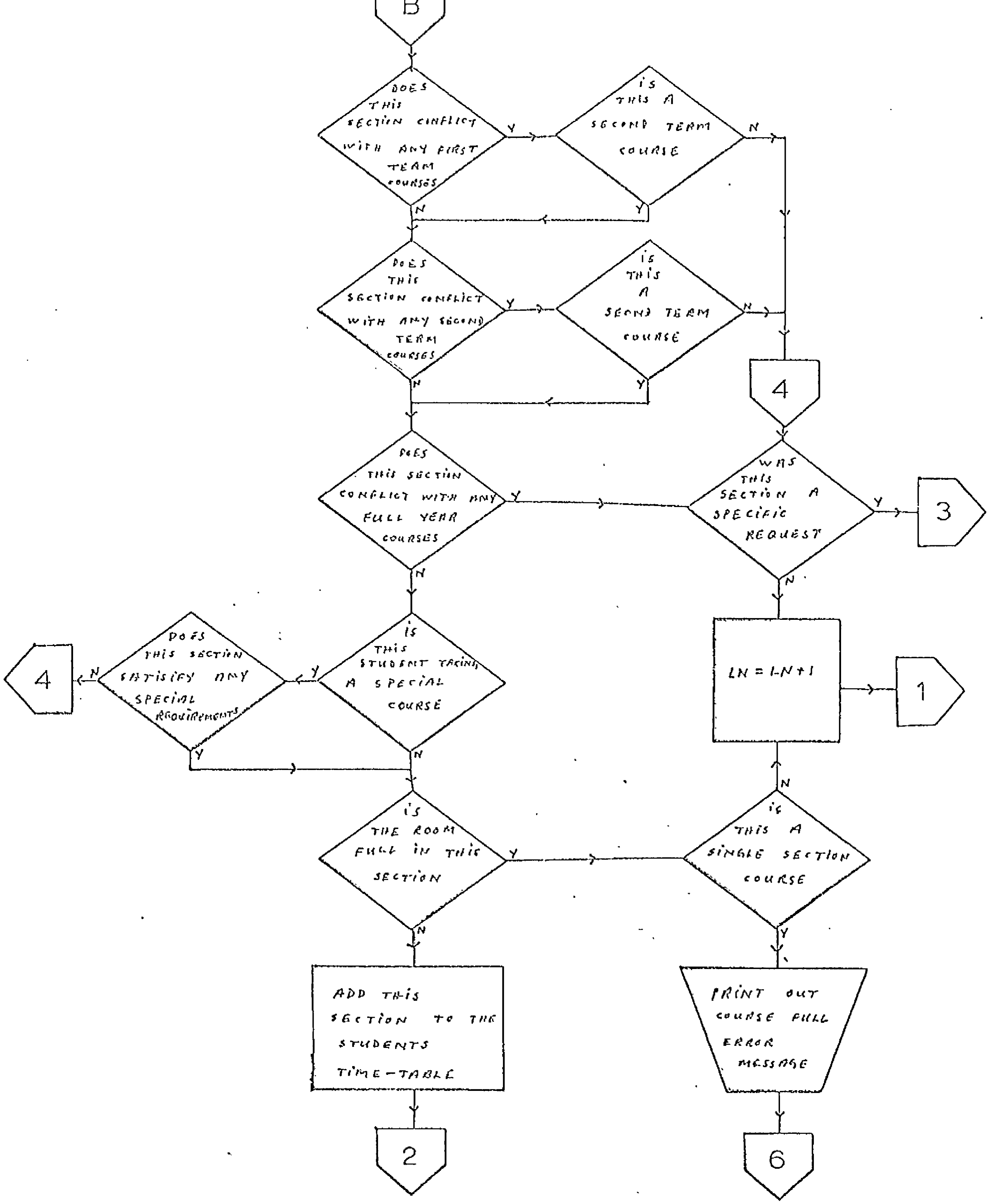

B
DOES THIS SECTION CONFLICT WITH ANY FIRST TERM COURSES
IS THIS A SECOND TERM COURSE
DOES THIS SECTION CONFLICT WITH ANY SECOND TERM COURSES
IS THIS A SECOND TERM COURSE
4
DOES THIS SECTION CONFLICT WITH ANY FULL YEAR COURSES
WAS THIS SECTION A SPECIFIC REQUEST
3
LN = LN + 1
1
IS THIS STUDENT TAKING A SPECIAL COURSE
DOES THIS SECTION SATISFY ANY SPECIAL REQUIREMENTS
4
IS THE ROOM FULL IN THIS SECTION
IS THIS A SINGLE SECTION COURSE
ADD THIS SECTION TO THE STUDENTS TIME-TABLE
PRINT OUT COURSE FULL ERROR MESSAGE
2
6
Y
N

FIGURE 2.2.2 (3)

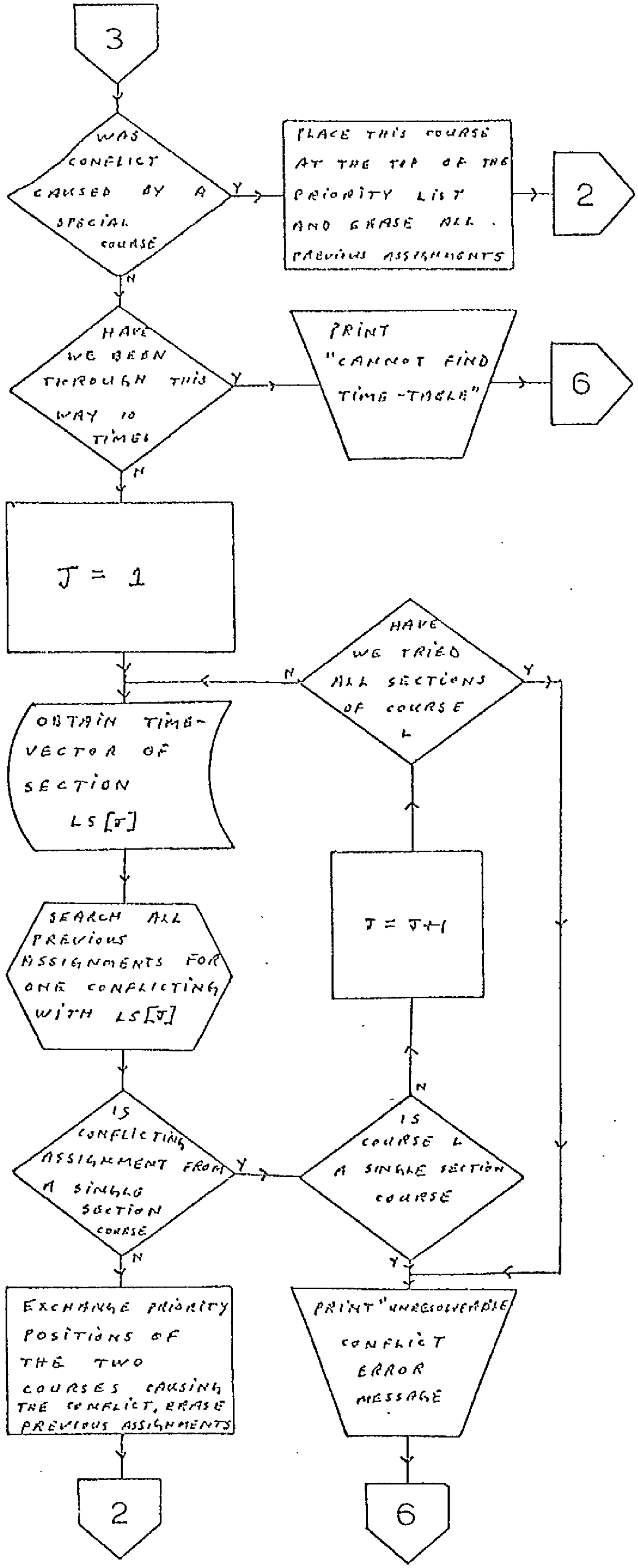

3
WAS CONFLICT CAUSED BY A SPECIAL COURSE
Y
N
PLACE THIS COURSE AT THE TOP OF THE PRIORITY LIST AND ERASE ALL PREVIOUS ASSIGNMENTS
2
HAVE WE BEEN THROUGH THIS WAY 10 TIMES
PRINT "CANNOT FIND TIME-TABLE"
6
J = 1
OBTAIN TIME-VECTOR OF SECTION LS[J]
HAVE WE TRIED ALL SECTIONS OF COURSE L
SEARCH ALL PREVIOUS ASSIGNMENTS FOR ONE CONFLICTING WITH LS[J]
J = J+1
IS CONFLICTING ASSIGNMENT FROM A SINGLE SECTION COURSE
IS COURSE L A SINGLE SECTION COURSE
EXCHANGE PRIORITY POSITIONS OF THE TWO COURSES CAUSING THE CONFLICT, ERASE PREVIOUS ASSIGNMENTS
PRINT "UNRESOLVEABLE CONFLICT ERROR MESSAGE
2
6

FIGURE 2.2.2 (4)

still left of what to do about it. In the first instance, the courses were selected for sectioning in ascending order of the number of sections available. By the previous assumption, that of an order existing for the selection of course requests, it appears than course M and course N were in the wrong order and that the first ordering heuristic, at least for this set of requests, was in error. The list of course requests is therefore rearranged, by interchanging course M and course N, so that the sectioning process, on its second try, will attempt to schedule course N before course M.

It should be noted that if a conflict is encountered on the second time through then a further "swap" on the two conflicting courses takes place. This process is advantageous only up to about ten swaps, the chance of a correct order being produced for a successful time-table diminishing rapidly thereafter, because of the course order in the REQUEST vector being reduced to virtual randomness.

The procedure used a SECTION vector to satisify the requirement of distributing the students evenly among the various sections of a course. When a course request is being considered a SECTION vector is produced such that the Ith element contains both the code number of the Ith section and the number of students previously assigned to that section. The SECTION vector is then sorted, by the number of students in each section, into ascending order. When the procedure attempts to find a section of the course into which the student might fit, it will try the section indicated by the first element of the SECTION vector, ie. the section with the lowest enrolment, and will only try the section with the greatest enrolment when all the others have been considered and rejected.

The section vector concept takes a very important part in two aspects of the procedure besides the section balance requirement. When a conflict has been detected and a swap has taken place, the second time through the procedure the elements of the section vector may be rearranged so that the same combination of sections is not retried. This best takes the form of a cyclic shift of the section vector's elements (of the course which originally caused the conflict) by one position, ie. take the first element, place it at the end of the vector and move all other elements up one position. In general if n swaps take place on this course, the first n elements of the section vector should be cyclically shifted. This double use of the section vector not only assures against the possibility of the procedure getting into a closed loop by trying the same section of the same courses each time, but also ensures that no section of a course fills to capacity before the other sections are within one or two places of themselves being full. Thus the student who is processed last has, on the whole, the same chance of being assigned to the "best" sections as the student who was processed first.

Another objective of the Colijn-Williams procedure is to satisfy the demand that a particular student is placed in a specific section, the constraint being sex, faculty or other personal/academic information. As a courses section vector is being compiled a check is made on a "reservation number" associated with each section, if this reservation number is non-zero it directs the procedure to a specific series of routines, the particular routine depending on the reservation number. These routines can deal with the section under consideration in three ways:

1/ Add this section to the section vector as a possible assignment, if the sex, faculty, or other information is consistent with what is required.

2/ Exclude this section from the section vector; this will, in effect, deny any knowledge of the existence of this section to the procedure and therefore render it impossible for the student to be assigned to it.

3/ Add this section to the section vector, and delete all other sections, thus forcing the student to be assigned to this section.

This technique was not restricted to the examination of a students personal data, and was used to examine the previous assignments in order to dynamically control the contents of the section vector. For example, if it is required that a student be assigned to the same section in both course I and course J, when course I is encountered during the assignment process, the reservation number assiciated with the course directs the program to the appropriate routine which checks to see if a section has been previously assigned for course J and, if so, forces the correct assignment for course I using point 3 above.

This system of heuristics has worked quite well. It was used for registering nearly 4000 students as they appeared for the start of the 1965 - 66 academic year at the University of Alberta, Calgary. 85% of the student body were successfully scheduled by the computer; of the rest of the students many had true conflicts due to their own carelessness or errors made during the data collection (a system involving mark sense cards). The greatest cause of error was the master time-table itself. Almost all of the second year chemistry students were rejected because of a conflict between two courses, both of which were compulsory for them; one half of the first year engineers were also rejected because the allowable enrolments in each section were only half of what they should have been.

Although the procedure itself was satisfactory the computer configuration on which it was implemented proved very limiting. The University administration required a very complex and voluminous output for each student processed, this combined with the slow access time to the disk unit resulted in the procedure taking about 30 seconds for each student.

These two slow peripheral devices were thus the limiting factor in the whole process. Very little can be done to speed up the output of the student's time-table except obtain a faster output medium, however some improvement can be obtained in the utilization of the disk unit (of course if the core store is large enough there need be no access to the disk, but few small universities have computers of this magnitude). To reduce the frequency of operations involving a random access device a procedure could be implemented to use the boolean matrix for the storage of the master time-table conflict data. The procedure would follow the same general lines as the time-vector scheme however when looking for a possible section assignment for a course the procedure would be able to "see" which sections were out of the question due to conflicts with previous assignments, rather than "grope around blindly" on the random access device to find the conflict free sections. This ability to "see" comes from the fact that, as each course is assigned a section, the row of the boolean matrix, corresponding to the section assigned, is added (by means of a boolean OR operation) to the accumulated rows of previous section assignments. It is then quite trivial to scan down this cumulative availability vector to determine the sections of the next course which are still available for use. This method would cut down the need to access the master time-table information from

$$2N + \sum_{i=1}^{N} S_i$$

accesses to about

$$2N$$

accesses in the case where no conflicts are encountered during the schedule completion (N is the number of courses requested by a student and S_i is the number of sections in the i th course request). When a conflict is encountered the number of accesses necessary to resolve the conflict will vary depending on the two courses in conflict and how much intermediate information is available in the core store.

Section 2.3 Transportation Networks

The relationship of sectioning to the problem of finding a flow through a network is interesting enough for a short dèscription. The problems involved in implementing this type of approach are formidable, particularly in the area of data storage. However the network flow approach to sectioning is interesting because it is able to determine whether or not a time-table exists for a particular student, and in so doing uses an algorithm which has been extensively studied by operations research personnel.

A transportation network is a finite directed graph, without loops, in which each arc is assigned an integer

$$C(x_i, x_j) \geq 0$$

known as the capacity of the arc from x_i to x_j, and in which:

1/ There is only one vertex x_0 such that all arcs joined to x_0 are directed away from x_0. x_0 is known as the entry to the network or as the source.

2/ There is one and only one vertex x_n such that all arcs joined to x_n are directed towards x_n. x_n is known as the exit from the network or sink.

A flow through a network is a function

$$f(x_i, x_j)$$

defined over all the arcs such that

$$f(x_i, x_j) \text{ is an integer}$$

$$0 \le f(x_i, x_j) \le C(x_i, x_j)$$

$$f(x_i, x_j) = f(x_j, x_i)$$

The value of the flow, F, is

$$F = \sum_i f(x_0, x_i) = \sum_i f(x_i, x_n).$$

One of the basic problems in network flows is to find the maximum F for a given network with a given set of capacities.

Of basic consideration in network flows is the idea of a cut. If α is a set of vertices of the network which includes x_n but not x_0, the set of arcs U_α^- connected terminally to α (ie. the arcs are directed toward α) is a cut of the network. For example in FIGURE 2.3.1

$$\alpha = x_n, x_2, x_3, x_6, x_9$$

$$U_\alpha^- = (x_8, x_n)\ (x_{10}, x_n)\ (x_0, x_2)\ (x_1, x_2)\ (x_5, x_2)$$
$$(x_0, x_3)\ (x_5, x_9)\ (x_{10}, x_9)$$

The cut is shown by a dotted line. This line is simply to show the terminally connected arcs which are

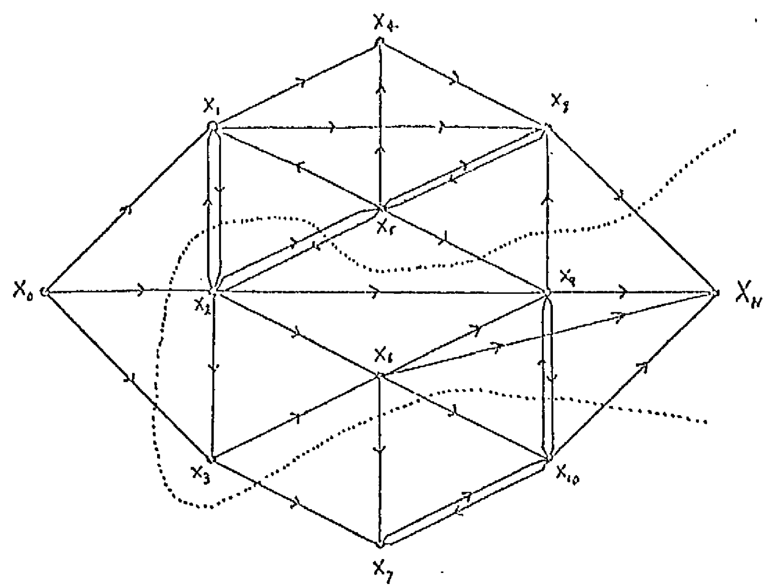


FIGURE 2.3.1

Showing a transportation network and a cut of this network encountered, it may also have nonconnected, open or closed segments depending on the choice of α.

Since α includes the sink, any flow from x_0 to x_n goes through at least one arc from U_α^-, thus whatever the flow F and the cut U_α^- may be

$$F \leq C(U_\alpha^-)$$

where $C(U_\alpha^-)$ is the sum of the capacity of the edges in the cut U_α^-.

If there is a flow F (from x_0 to x_n) and a cut V such that

$$F = C(V)$$

the flow is a maximum and the cut is of minimum capacity; this is essentially the theorem proved by Ford and Fulkerson (14)

> In a given transport network the maximal flow is equal to the minimal cut.

To represent the sectioning problem it is possible to define a transport network as follows

- a source x_0 and a sink x_n
- to each course i offered by an institution let there be two vertices x_i and y_i
- the existing arcs and their capacities are defined as follows (where R is the set of courses requested by a student and $|R|$ is

the number of courses requested)

$$C(x_0, x_i) = \begin{cases} |R| - 1 & (\text{if } i \in R) \\ 0 \text{ otherwise} \end{cases}$$

$$C(x_i, y_j) = \begin{cases} 1 \text{ if } i \text{ is not given at the same time as } j, \text{ and } i \in R,\ j \in R \\ 0 \text{ otherwise} \end{cases}$$

$$C(y_j, x_n) = \begin{cases} |R| - 1 & (\text{if } j \in R) \\ 0 \text{ otherwise} \end{cases}$$

If a time-table can be found for this student then this network must have a maximum flow, F, such that

$$F = |R|^2 - |R|$$

For example consider the flow out of x_0, there are $|R|$ courses and each arc, by construction, has a flow of $|R| - 1$, thus the total flow out of x_0 is $|R|^2 - |R|$. Similarly the flow into x_n can have a maximum value of $|R|^2 - |R|$. For every $x_i \in R$ the flow out of this vertex must be equal to the flow into it, ie. $|R| - 1$. By construction

$$C(x_i, y_j) = 1$$

if course i and course j may be taken simultaneously. Again, assuming a time-table is possible,

$$f(x_i, y_j) = |R| - 1 \quad (\text{for } i, j \in R)$$

and

$$\sum f(x_i, x_j) = |R|^2 - |R|$$

so that the cut containing only those vertices in R must be such that

$$C(V) = F = |R|^2 - |R|.$$

Application of the Ford-Fulkerson algorithm (23) will show if any network has a maximal flow of $|R|^2 - |R|$ and thus show the existence, or nonexistence, of a time-table for the student under consideration.

This same concept may be extended to the situation where each course consists of several sections. By dividing the course sections into time disjoint groups it is possible to define a series of k sets D_1, D_2, $D_3, \ldots, D_k$, each set containing those sections given at the same or overlapping time periods. A transportation network may now be set up with the vertices as follows

- x_0 source
- x_n sink
- x_i for each course i
- x_{ij} for each section j of course i
- D_l for each time division l

whose arcs and their capacities are defined as follows

$$C(x_0, x_i) = \begin{cases} 1 & \text{for all } i \in R \\ 0 & \text{otherwise} \end{cases}$$

$$C(x_i, x_{ij}) = \begin{cases} 1 & \text{if } j \text{ is a section of course } i \\ 0 & \text{otherwise} \end{cases}$$

$$C(x_{ij}, D_l) = \begin{cases} 1 \text{ if } j \text{ is offered in time division } l \\ 0 \text{ otherwise} \end{cases}$$

$$C(D_l, x_n) = 1 \text{ for all } D_l$$

If a workable time-table is possible then there exists a set of $|R|$ sections, one for each course in R, which are joined to $|R|$ different D_l.

The maximal flow out of x_0 can be seen to be $|R|$. For the flow out of x_i to equal the flow into x_i there must exist only one x_{ij} for each x_i (because the flow must be integral valued) to retain the maximal flow at its value of $|R|$. For the maximal flow into x_n to be $|R|$ the flow must have come from $|R|$ different D_l because by construction

$$C(D_l, x_n) = 1.$$

Therefore the maximum flow must proceed from the $|R|$ separate x_{ij} to the $|R|$ separate D_l, thus the network is shown to have a maximal flow of

$$F = |R|$$

when a time-table exists and a flow of

$$F < |R|$$

when a time-table does not exist. Nonexistence of a time-table will be shown by arcs with the required capacity of $|R|$ not being present from x_{ij} to D_l, or if they are, then arcs from two selected x_{ij} will be incident with one D_1 and because

$$f(D_1, x_n) = 1$$

the flow will be $|R| - 1$ or less.

The Ford-Fulkerston algorithm for finding the maximal flow through a network reouires that the path of this flow be traced and from this trace come the nonconflicting course sections satisfing R.

This is a very satisfying formulation and solution to the sectioning problem as the procedure used has been well investigated and the theory (very little of which has been mentioned here) has been extensively studied.

Section 2.4 Complete Graph Algorithmic Approach

The major problem with an heuristic sectioning procedure is to determine if a set of conflict free assignments exists without trying all possible sections. In general, the situations in which a time-table does exist are quite easily solved by heuristic procedures, however in the situations in which time-tables do not exist the computer spends a great deal of time in the assignment and back-tracking sections of the procedure, and is generally forced to give up without either finding a time-table or determining that one does not exist. Some aspects of the heuristic procedures, such as the concept of a section vector, are well worth keeping but the basic assignment and backtracking loops need to be modified.

P. Hall's famous dissertation "On the Representatives of Subsets" (18) provides a method of determining whether a student's course requests are compatible. Hall states (in his Theorem 2):

> Given any set S (divided into any number of classes $S_1, S_2, \ldots, S_n$) and a finite system of subsets of S ($T_1, T_2, \ldots, T_m$) such that
>
> $$S_i \wedge S_j = \emptyset \qquad (\text{for } i \neq j)$$
>
> then there always exists a set of m elements

(a_1, a_2, ..., a_m) no two of which belong to the same class, such that

$$a_i \in T_i \qquad (i = 1,2,3,\ldots,m)$$

provided only that for each $K = 1,2,3,\ldots,m$ any K of the sets T_i contain between them elements from at least K classes.

The proof of this statement is in the original journal, however the arguments used in the section on network flows (Section 2.3) may be used as a proof.

The set S may be considered as the set of sections of the m requested courses of a student. The class S_i are sections meeting at time period i. The subsets of S (T_1, T_2, ..., T_m) are the sections of the m individual course requests. The set of m distinct representative elements now correspond to the section of each class which would provide the nonconflicting time-table ($K = m$).

It is only necessary to verify the sufficiency condition of Hall's theorem for each student's course requests to determine the existence, or nonexistence of a time-table. For nontrivial cases this verification may require a large amount of computing, thus it would be advantageous to produce, as a byproduct of the verification, the actual student time-table.

This verification can be accomplished with the procedures developed to find complete subgraphs in

Section 1.5. First, however, the master time-table must be stored in the form of a boolean matrix. To each section of each course there corresponds one row and column of the time-table boolean matrix, T, with the requirement that t_{ij} is true if section i is given at the same, or overlapping, time as section j. With the master time-table in this form the verification of Hall's sufficiency condition reduces to the following steps.

1/ For each student construct a boolean matrix, M, consisting of only those rows and columns of T corresponding to the sections of his requested courses (this matrix is associated with a graph G which is a subgraph of T).

2/ If any row, j, of M represents a single sectioned course then scan this row for any true element, i, and eliminate the row and column i from the matrix. This elimination may take the form of setting all elements in row and column i to the value true but preferably should consist of actually removing row and column i from M as this will speed up the later stages of the algorithm. This elimination process corresponds to an indication that there is no alternative section assignment available for course j.

3/ Ensure that there is still at least one section available for assignment in each requested course. If all sections of any course were deleted in step 2 then it will be impossible to construct a time-table for this student.

4/ Generate edges between vertices corresponding to the individual sections of each course request, ie. if a course is divided into s sections (1,2,3,...,s) then M_{ij} should be set to the value true for all i and j = 1,2,...,s (including the case i = j). This will ensure that this student is not placed in two sections of the same course.

5/ Take the complement, $\overline{M}$, of the matrix M ($\overline{M} = 1 - M$). $\overline{M}$ is now a boolean matrix corresponding to a graph, $\overline{G}$, of the pairwise permissible course sections.

6/ If the student has requested N courses then, using the complete graph algorithm developed in Section 1.5, determine the complete subgraphs of order N in $\overline{G}$.

7/ If any complete subgraphs of order N exist in $\overline{G}$ then Hall's sufficiency condition has been obtained and a time-table corresponds to the N vertices in any K_N of $\overline{G}$.

Step 1 produces a boolean matrix which indicates which section of the $|R|$ requested courses may not be taken simultaneously. For example FIGURE 2.4.1 shows the graph, and its associated matrix M, of the conflicts between the sections of five courses:

A - a single sectioned course (vertex 1)

B - a single section course (vertex 2)

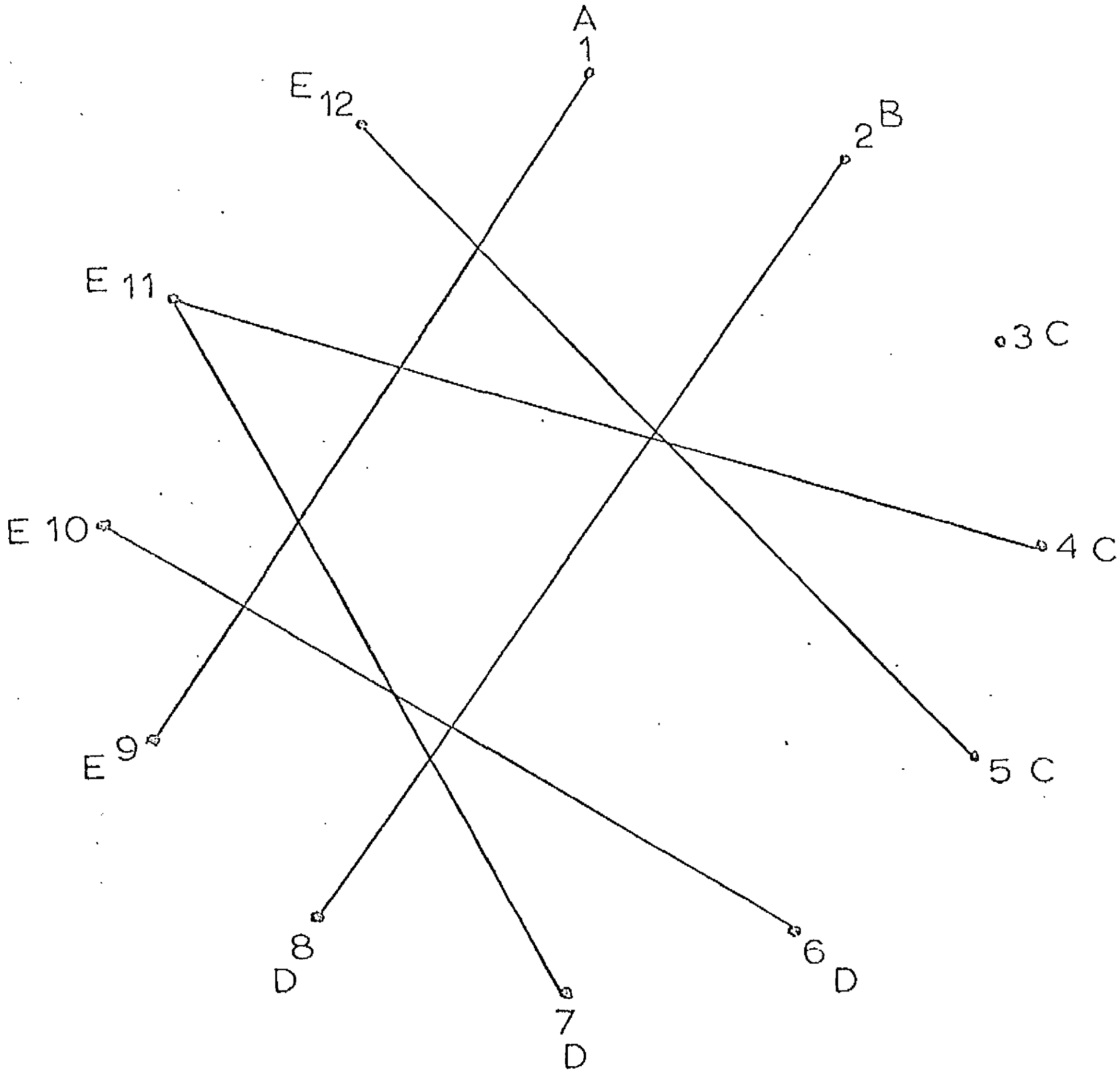
C - a three section course (vertices 3, 4, 5)

D - a three section course (vertices 6, 7, 8)

E - a four section course (vertices 9, 10, 11, 12).

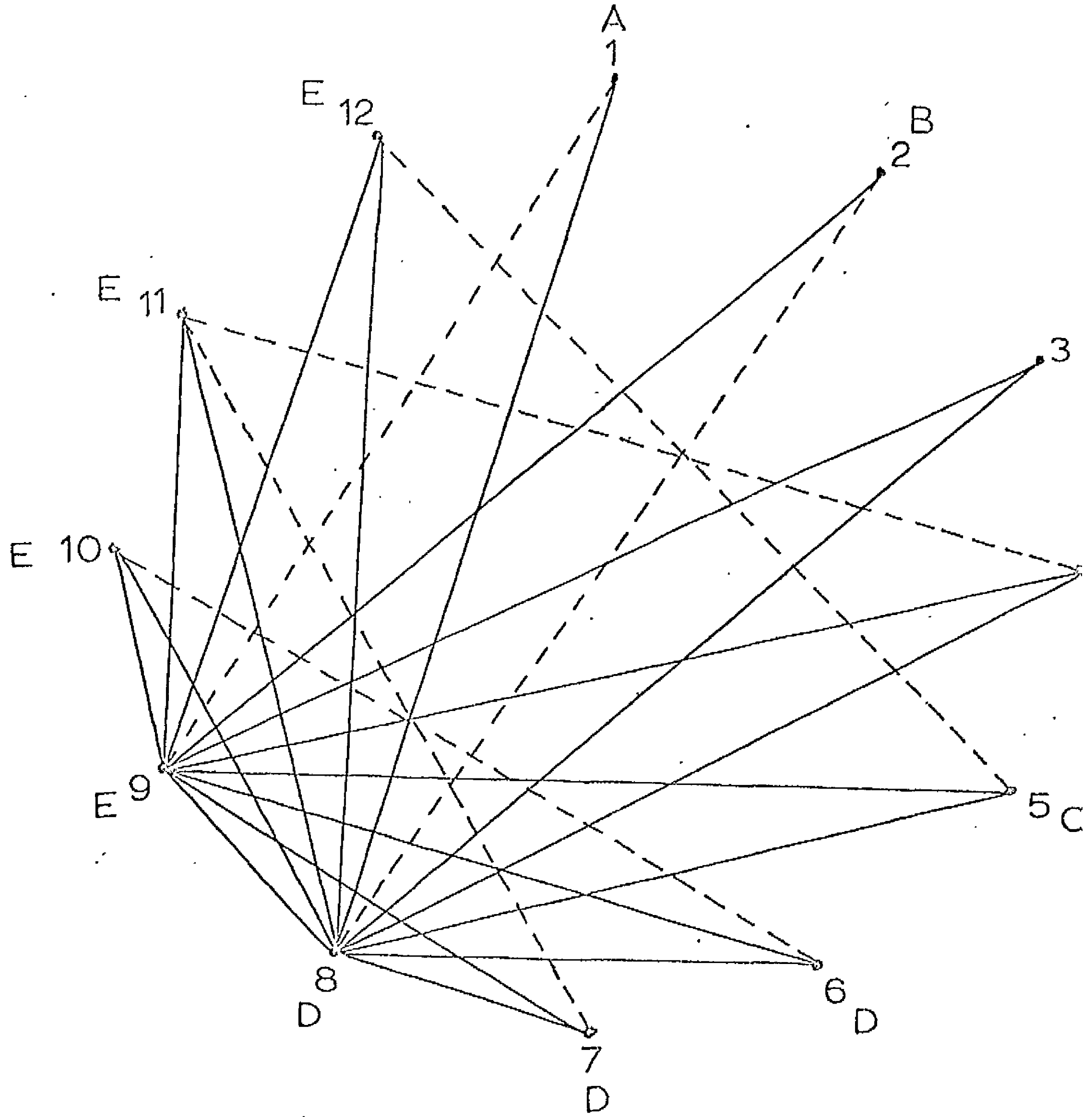
From FIGURE 2.4.1 it can be seen that section 1 (course A) is given the same time as section 9 (the first section of course E) and section 2 (course B) is given at the same time as section 8 (the third section of course D). Because courses A and B have only one section each it is obvious that sections 8 and 9 cannot possibly be included in any time-table with 1 and 2. Step 2 takes care of this incompatibility by effectively removing these two vertices (8 and 9) from the graph by generating edges between them and all other vertices. This step is illustrated in FIGURE 2.4.2, the new edges being the continuous lines and the original edges the dashed lines.

Step 4 takes care of the possibility that a student may be assigned to two sections of the same course. It is easily done by generating edges between



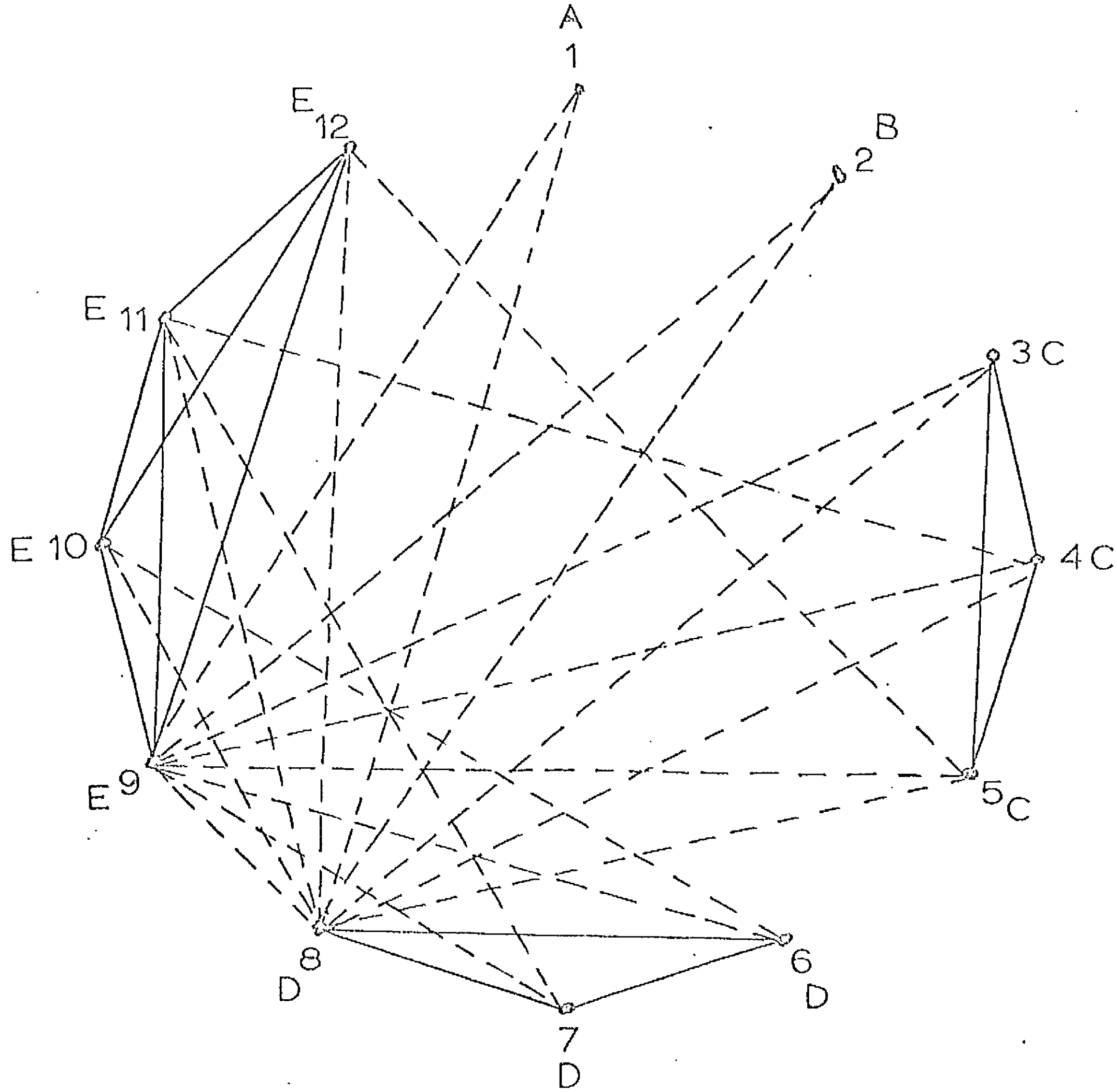
$$M = \begin{array}{r|cccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
1 & & & & & & & & & 1 & & & \\
2 & & & & & & & & 1 & & & & \\
3 & & & & & & & & & & & & \\
4 & & & & & & & & & & & 1 & \\
5 & & & & & & & & & & & & 1 \\
6 & & & & & & & & & & 1 & & \\
7 & & & & & & & & & & & 1 & \\
8 & & 1 & & & & & & & & & & \\
9 & 1 & & & & & & & & & & & \\
10 & & & & & & 1 & & & & & & \\
11 & & & & 1 & & & 1 & & & & & \\
12 & & & & & 1 & & & & & & &
\end{array}$$

FIGURE 2.4.1



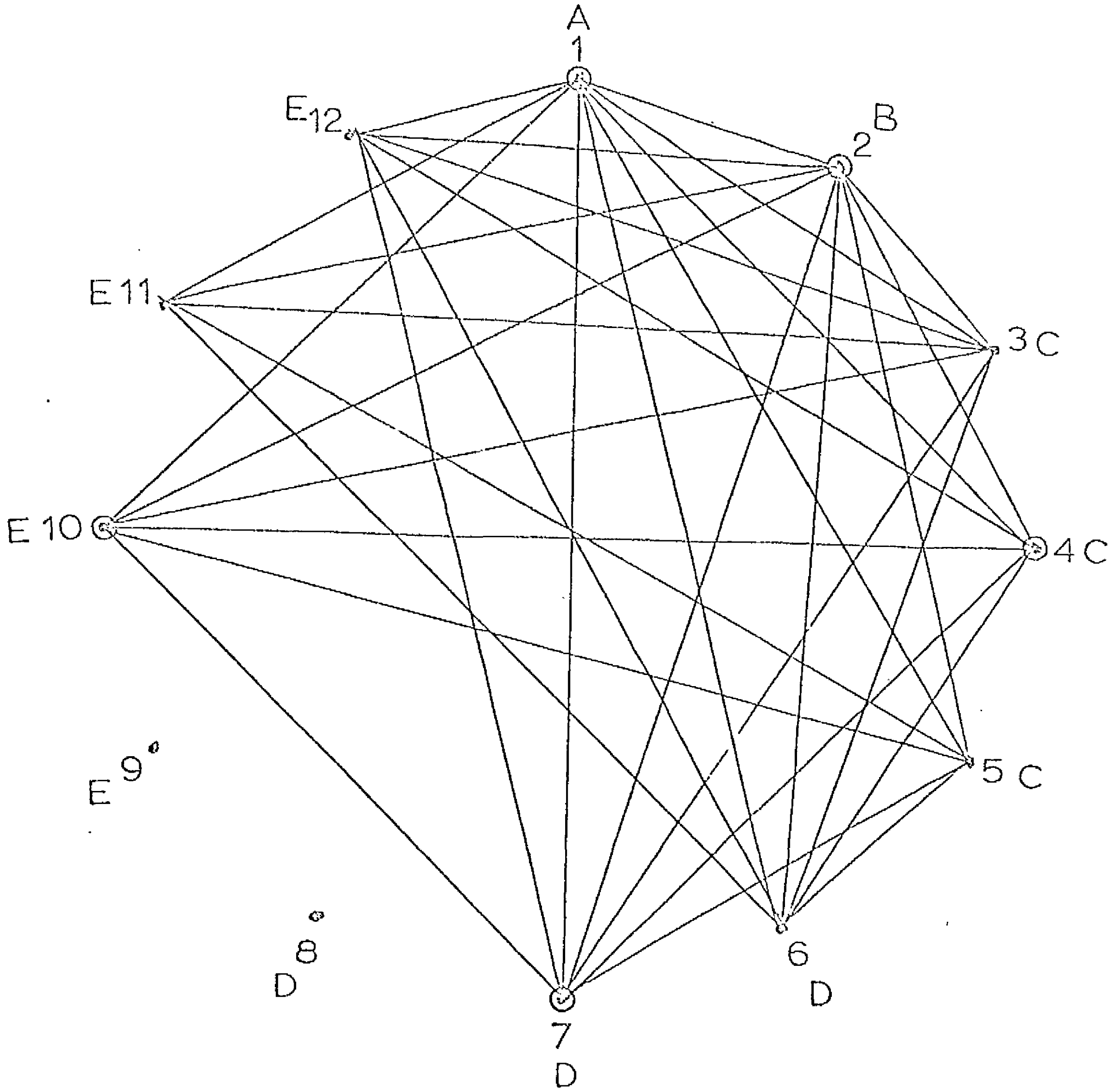
$$M = \begin{array}{c|cccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
1 & & & & & & & & 1 & 1 & & & \\
2 & & & & & & & & 1 & 1 & & & \\
3 & & & & & & & & 1 & 1 & & & \\
4 & & & & & & & & 1 & 1 & & 1 & \\
5 & & & & & & & & 1 & 1 & & & 1 \\
6 & & & & & & & & 1 & 1 & 1 & & \\
7 & & & & & & & & 1 & 1 & & 1 & \\
8 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
10 & & & & & & 1 & & 1 & 1 & & & \\
11 & & & & 1 & & & 1 & 1 & 1 & & & \\
12 & & & & & 1 & & & 1 & 1 & & &
\end{array}$$

FIGURE 2.4.2



$$M = \begin{array}{r|cccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
1 & 1 & & & & & & & 1 & 1 & & & \\
2 & & 1 & & & & & & 1 & 1 & & & \\
3 & & & 1 & 1 & 1 & & & 1 & 1 & & & \\
4 & & & 1 & 1 & 1 & & & 1 & 1 & & 1 & \\
5 & & & 1 & 1 & 1 & & & 1 & 1 & & & 1 \\
6 & & & & & & 1 & 1 & 1 & 1 & 1 & & \\
7 & & & & & & 1 & 1 & 1 & 1 & & 1 & \\
8 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
9 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
10 & & & & & & 1 & & 1 & 1 & 1 & 1 & 1 \\
11 & & & & 1 & & & 1 & 1 & 1 & 1 & 1 & 1 \\
12 & & & & & 1 & & & 1 & 1 & 1 & 1 & 1
\end{array}$$

FIGURE 2.4.3



$$
M = \begin{array}{r|cccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\
\hline
1 & & 1 & 1 & 1 & 1 & 1 & 1 & & & 1 & 1 & 1 \\
2 & 1 & & 1 & 1 & 1 & 1 & 1 & & & 1 & 1 & 1 \\
3 & 1 & 1 & & & & 1 & 1 & & & 1 & 1 & 1 \\
4 & 1 & 1 & & & & 1 & 1 & & & 1 & & 1 \\
5 & 1 & 1 & & & & 1 & 1 & & & 1 & 1 & \\
6 & 1 & 1 & 1 & 1 & 1 & & & & & & 1 & 1 \\
7 & 1 & 1 & 1 & 1 & 1 & & & & & 1 & & 1 \\
8 & & & & & & & & & & & & \\
9 & & & & & & & & & & & & \\
10 & 1 & 1 & 1 & 1 & 1 & & 1 & & & & & \\
11 & 1 & 1 & 1 & & 1 & 1 & & & & & & \\
12 & 1 & 1 & 1 & 1 & & 1 & 1 & & & & &
\end{array}
$$

FIGURE 2.4.4

the sections of a course, so that if one section is assigned to a student all other sections in that course are then incompatible with the time-table. This is shown in FIGURE 2.4.3, with the newly inserted edges again being shown as continuous lines and the previous edges as dashed lines.

Step five produces the complementary graph, that is the graph with the same number of vertices but where the original graph had an edge the complement has none, and where the original graph did not have an edge the complement has. The complementary graph now indicates which pairs of courses are permitted together in a time-table. It is now only neccssary to find $|R|$ (in this case 5) vertices forming a complete graph to both verify the sufficiency condition of Hall's theorem and find a time-table. FIGURE 2.4.4 illustrates the complementary graph and the circled vertices one of the many complete five graphs available for a time-table.

The method of constructing this graph ensures that there will be no complete graphs of order $|R| + 1$, and only a complete graph of order $|R|$ if a time-table exists. Thus the operation $\Delta T_{|R|}$ (see section 1.4) performed on the matrix $\overline{M}$ will quickly determine if a time-table is possible, and only if it is, is it necessary to continue on to find the complete graphs which actually represent the time-tables.

The amount of computation necessary to perform the operation ΔT_n is dependent on $\frac{v^2}{2}$ where v is the number of vertices in the graph. For this reason it is best, in step 2, to completely remove a row and column from the matrix M rather than adding the extra edges. Which of these two methods are actually used will be heavily dependent on the computer in use and the method of storing the matrix M.

In very bad cases the matrix M may be larger than 100 X 100. This, because of the amount of computation necessary for the operation ΔT_n, may lead to excessive computer time being used, however this may be alleviated if the computer has a range of powerful logical instructior The general availability of machines with multiple processing units will also aid in overcoming the problem of excessive computer time being used. The operation ΔT_n is suited for multiple processor machines because the checking of the number of edge circuits of length three subtended on each edge is independent of operations on other edges. Although it can be shown that serial processing may result in fewer actual operations being performed, the time saved by parallel processing will be significant in the real time situations in which student sectioning is usually carried out.

Section 2.5 Experience With the Complete Graph Sectioning Algorithm

The complete graph sectioning algorithm was implemented in an English-Electric-Leo-Marconi KDF 9 using data obtained from the Registrar of the University of Alberta, Calgary. This implementation was not intended for an actual sectioning production run, as was the heurictic implemented in Section 2.2, but simply to gain experience in the running problems of this algorithm. The implementation consisted of four separate programs:

1/ A program to read the master time-table data from cards and produce a type 2 time-vector - written in ALGOL with some procedure bodies in USER CODE

2/ A program to read the type 2 time-vectors and from them build up the master time-table boolean matrix which was then stored on the KDF 9 disk unit - written in ALGOL with some procedure bodies in USER CODE

3/ A program to read students course requests and produce the individual boolean matrices by selecting the appropriate rows and columns from the master time-table matrix - written in ALGOL with some procedure bodies in

USER CODE

4/ The actual sectioning algorithm - written within the framework of ALGOL but most of the program is written in USER CODE in order to obtain full use of the KDF 9 logical instructions and to simulate as closely as possible actual production conditions.

Programs 1, 2, and 3 were not written with the intent that they be as fast as possible. In fact it turned out that, due to the access time on the disk unit, better usage of the computer time would have resulted if program 2 had never been written - the individual student matrices being built up by direct comparison of the time-vectors rather than removing the relevant rows and columns from the disk. Program 4, on the other hand, was written with the intention that its running time should be kept to a minimum.

By a slight modification of the complete graph procedure it is possible to cause it to find all the complete graphs of a given order rather than just one complete graph. This was used in an attempt to produce all possible time-tables for a student, thus allowing a selection procedure based on the "goodness" of a particular time-table. It was found that, after the first complete graph had been found, the subsequent complete graphs were found at a rate which was limited only by the output devices (magnetic tapes with a transfer rate of 40,000 characters per second). Thus it seems possible that a number of different time-tables can be completed and the "best" one given to the student.

During an attempt to find a function which would predict the amount of computing time each student would need, a plot of the number of vertices in the individual student's conflict graph against the computer time used was produced (see FIGURE 2.5.1). This showed an alarming tendency for the graphs to fall into three distinct types. Type A (see FIGURE 2.5.1) is easily explained as the graphs of students whose course requests were such that they did not possess a conflict-free time-table. Type B and C however are students who did possess a time-table and no simple test could detect the difference between a type B and a type C graph. Plots of both the average number of sections in each course against time, and the average degree of each vertex against time showed this same threefold division.

It became imperative to find the cause of this division when it was found that one student required 1231 seconds computing time to determine a time-table. This student's final graph was large, 110 vertices with 5588 edges, and was so constructed that it had a possible 14,929,920 different section combinations. However the simple size of the graph had to be disregarded when

Computing time taken with
the complete graph sectioning
procedure

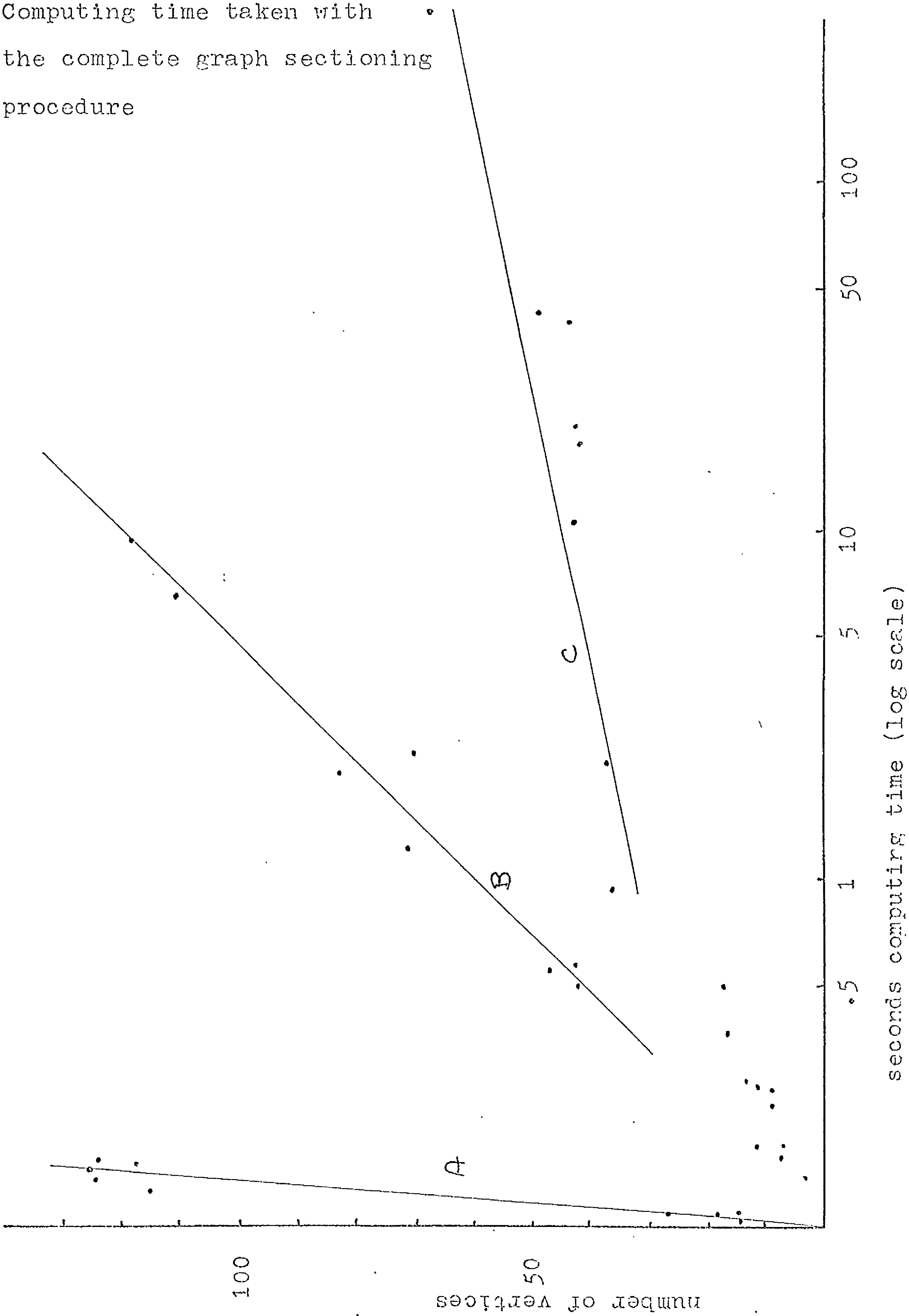


FIGURE 2.5.1

another student was found with the same size of graph (110 vertices, 7234 edges, and 218,350,080 possible combinations of sections - about 20 times as many as the first student) whose time-table only took 6.6 seconds to compute.

As it turned out the basic nature of the complete graph algorithm, a yo-yo tree search, was the culprit. If, in searching down the tree, the algorithm initially chooses an unproductive branch, a great deal of effort is wasted in searching all the offshoots of this branch before the procedure can again look for a more fruitful branch. In the case of the two students cited above, the former made the procedure search seven long unproductive branches before it found one leading to a complete graph, in the case of the latter student the procedure found a fruitful branch immediately and after only recursing six times it found the complete graph.

Any attempt to eliminate the searching of unproductive branches must eliminate the ability of the algorithm to find all possible time-tables for a student. However computing times of 1231 seconds for a single student are also unacceptable. A compromise must be found.

Each step down the complete graph search tree eliminates from the graph at least one vertex which does not possess the necessary edges to form a complete graph. This elimination means that, as one goes further down a direct fruitful branch, the ratio, β, where

$$\beta = \frac{\text{number of edges in this graph of } N \text{ vertices}}{\text{number of edges in a } K_N} \qquad (2.5.1)$$

must increase. Thus if β is computed for each step, and compared to the ratio obtained in the last step, an indication of the possible "fruitfulness" of the branch is obtained.

When this extra step was incorporated into the complete graph sectioning procedure the case which formally required 1231 seconds computing time now required only 23 seconds. As can be seen from FIGURE 2.5, this added step tended to bring the B and C types of graph together and very substantially lowered the total computing time necessary, although the computing time for each of the type B graphs was slightly increased.

This step is detrimental to the algorithm as it may prevent it finding all the complete graphs. It is however a necessary step if sectioning is to be done in a real time situation.

FIGURE 2.5.3 shows the behaviour of the search for several typical students. The RATIO 2.5.1 (shown as a percentage) is plotted against each recursive step (or branch point in the search tree) met during the search, the height of the plot indicates the completeness of the subgraph. The original type of each graph

Computing time taken with complete graph sectioning procedure using the β ratio

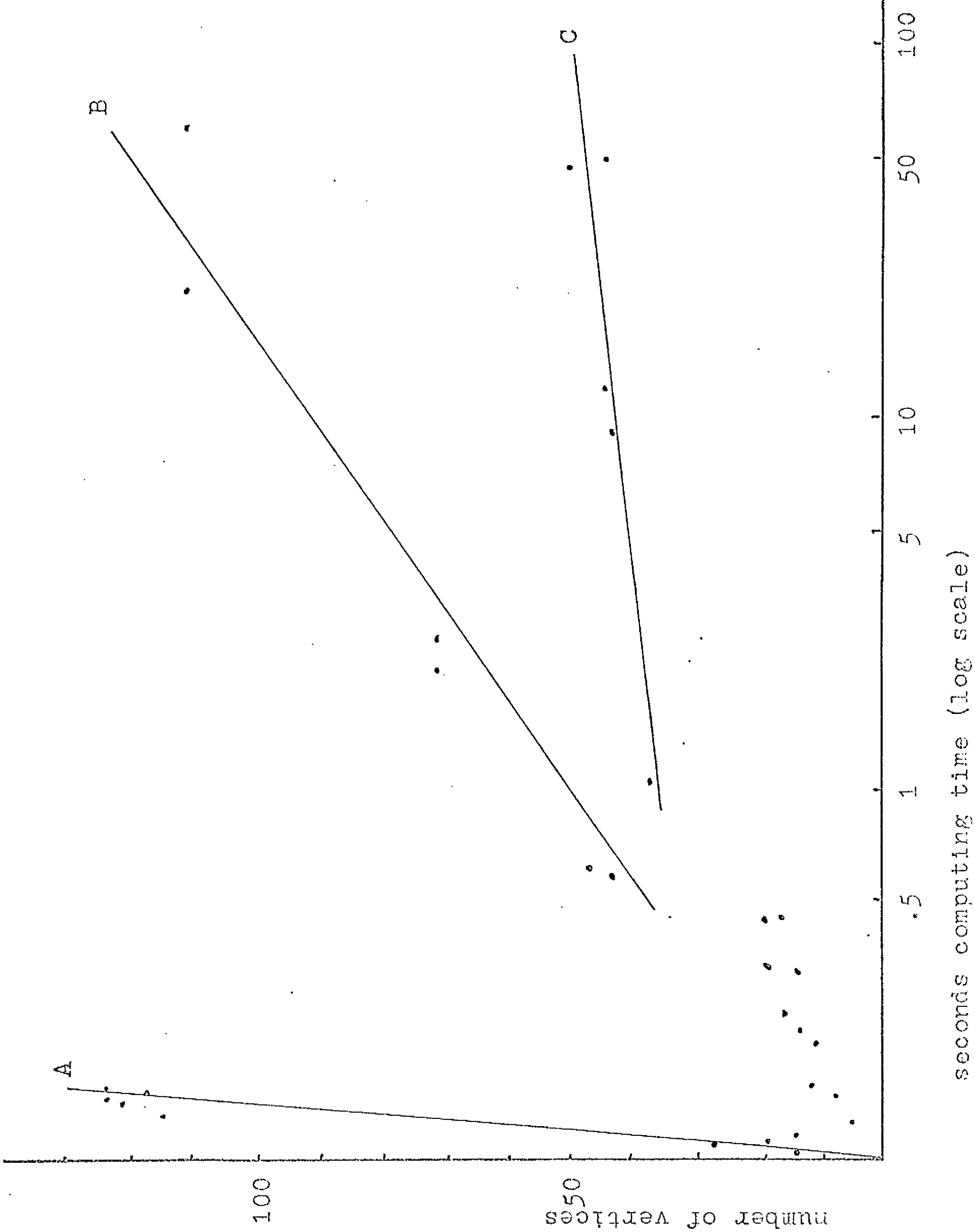


FIGURE 2.5.2

A plot of β against the number of steps taken in determining a complete subgraph

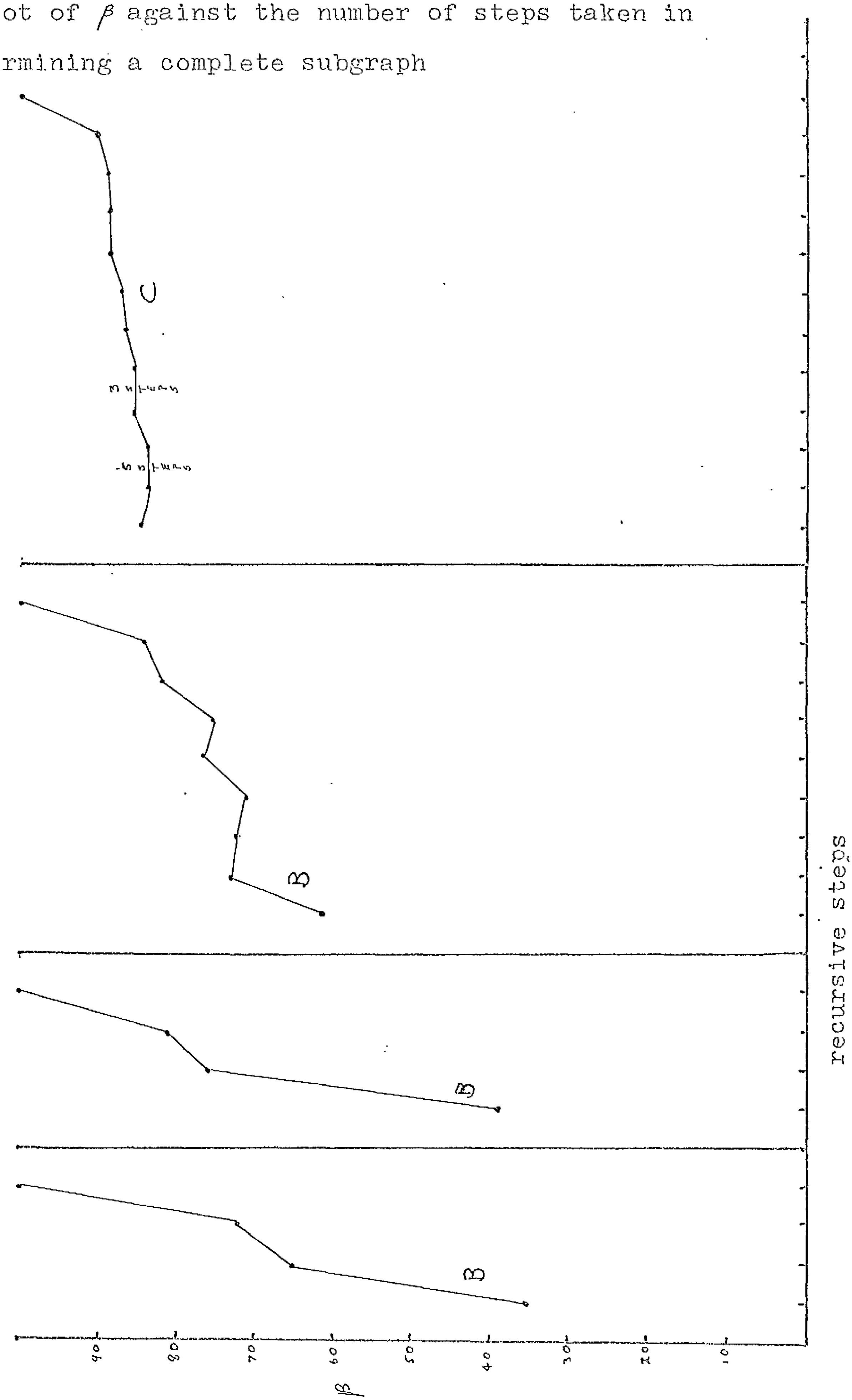


FIGURE 2.5.3

(from line B or C on FIGURE 2.5.1) is indicated beside the plot.

The processing time varied from 0.3 seconds to 56 seconds per student with an average of less than 10 seconds, well within the time allowable for a universit of moderate size with a computer the size of KDF 9.

C H A P T E R 3

Master Class-Teacher-Room Time-tables

Section 3.1 The Problem

This final aspect of the three part time-table problem is the most complex (and as a result least understood) of the time-tabling situations arising in educational administrations. The construction of a master time-table may be considered as the next logical step after the student sectioning problem has been solved. However this involves progressing from a one dimensional scheduling problem (examination time-tables) where optimality can be closly defined and virtually attained, to a two dimensional scheduling problem (student sectioning) where a definition of optimality can only be vaguely suggested, to a four dimensional scheduling problem where optimality is practically (if not actually) impossible to define and to simply find a feasible solution would be considered an achievement. The fact that the problem has reached into four dimensions does not, of itself, prove the stumbling block, but the subtle interplay of the constraints to the variables (possible free selection of courses by students, ensuring departments are not overloaded in any one teaching hour, limited classroom availability, preferences of faculty members for certair times of day, the possibility of several courses being given by the same person - to mention but a few) are the factors which make it remarkable that master time-tables exist at all, let alone attempting to define an optimum solution.

Producing a master time-table for a university becomes an intricate problem particularly if it is attempted to give each student the fullest possible choice of subjects. Even carefully designed master time-tables require a large number of students to take courses they would not have chosen except for the fact that "It was the only one that would fit into my schedule".

The ideal situation would be to allow students to register, then use a computer to produce a master time-table from data obtained during the registration. Aside from the fact that a student would be free to register for any course he is otherwise qualified to take, this method of producing a master time-table may lead to a considerable improvement in the utilization of an institution's physical facilities. The traditional schedule is to group the lectures in the mornings and leave the afternoons free for the longer laboratory and tutorial sessions. It is, of course, very wasteful to have laboratories (probably the most expensive class room space in a university) idle for one half of each day, but to schedule a three hour chemistry laboratory for the morning is to court possible disaster in the individual student's schedule. Having the master time-table produced after registration might well lead to some of the laboratory periods being scheduled in the morning with some of the lectures in the afternoon. On the other hand it may be quite impossible to produce a feasible master time-table by this method and some limitations may still have to be placed on the course combinations selected by the students.

Section 3.2 Proposed Procedures

The literature available on the production of master time-tables may be divided into four distinct groups characterized by their approach to the problem. The four groups are 1/ mathematical, 2/ clerical, 3/ algorithmic, and 4/ heuristic. It will suffice to describe one example from each category as being typical.

G. R. Sherman (31,32) has published what is perhaps the most comprehensive work which attempts to define the problem from a purely mathematical basis. Using set theory and probability distributions of students selecting various sets of courses he has managed to formally define the various steps necessary in the solution of the problem as well as some of the relations which must hold true for the resultant schedule to be actually implemented. In Sherman's major work (32) an attempt has also been made to define what is meant by a "good" schedule. This, unfortunately, is taken from the view that "good" only applies to each set of resources, rather than the institution as a whole.

Although Sherman has developed algorithms to perform the various steps he defines, the author was unable to find any record of them actually being implemented in a realistic situation. This undoubtedly stems from the fact that they are almost purely combinatorial in nature and would be very extravagant, if not impractical, in the computer time used. It is also rather unfortunate that his work, being a set theory dissertation, is extremely difficult to read and thus his potentially useful definitions are incomprehensible to the vast majority of administrative personnel who would be able to benefit from them.

The second, and perhaps most fruitful, approach is typified by the system known as G.A.S.P. (Generalized Academic Simulation Programs) devised by R. E. Holtz (21) who worked under the direction of the Registrar's Office at the Massachusetts Institute of Technology. The basic philosophy behind the system is:

> "As scheduling involves many highly responsible personnel, and considerable clerical work is involved in the decisions to be made, you have high level people simply making a few decisions and then doing mountains of paper work. G.A.S.P. programs were designed to be used by the persons charged with building the schedule; the computer simply taking over their role as clerks."

According to the published results a registrar, starting from basic data, will have a workable time-table after 3 - 4 runs, and after 10 - 20 runs, spaced a day or so apart, he would have a master time-table better than any he could have produced by traditional methods. After each successive run the good features of a time-table are noted and the bad features are modified by the user. Thus proposed schedule innovations can be studied for feasibility much more readily than is possible with manual procedures. For example, if, during a series of runs, the data on students, times, and staff are kept fixed while the number and size of classrooms are varied then the schedules resulting from the series of runs would give a valuable insight to the number of class rooms actually required. This same method would prove a valuable tool in forecasting the requirements of the institution in the future.

The G.A.S.P. scheduling system is designed as a four dimensional assignment problem (time, rooms, staff, and students). Its approach is to sacrifice an exact, conflict free solution in return for keeping the ability to make all four of the assignments and to be of use to very large institutions. The time-table construction routines will schedule classes in the order designated by the user, time assignments being based on the availability, influenced by a user generated weighting factor, of staff rooms, and students. If the program is unable to make a required assignment, from its specified choices, it simply leaves that job for the user to do manually or for a future run when the user has altered the input data.

The G.A.S.P. system is composed of several programs only one of which is concerned with the actual construction of a master time-table. The other programs will attempt to schedule a representative group of students to the new master time-table and output various statistics to aid the user in evaluating the resulting time-table.

The scheme proposed by C. C. Gotlieb and J. Csima (7,17) for the solution of school time-tables envisages the construction of the three dimensional boolean array B_{ijk} each element of which represents the meeting of a class (i) with a teacher (j) at a particular hour (k). A <u>false</u> element indicates that this class is not available to meet with this teacher at this hour. Initially the array is filled with the value <u>true</u>, indicating that any teacher is available to meet with any class at any hour. The procedure then modifies the array so that, at the conclusion of the modification, at each hour it is possible for each teacher to meet only one class and for each class to meet only one teacher, and each teacher can meet each class a predetermined number of times. The time-table is then inherent in the resulting array.

Gotlieb ensures that the resulting time-table conforms to certain desirable patterns and fully exploits facilities in heavy demand by allowing preassignments to be made. Although he has been able to prove that the procedure will detect when a time-table is impossible under a given set of preassignments, he has not been able to show that, for a given set of classrooms, teacher, and times, a time-table exists. He has, however been able to prove existence in some special cases.

The procedure requires that, at regular intervals during the procedure, an examination be made of each plane section of the three dimensional array. It is shown that each plane section is effectively square in that any seeming excess of rows or columns can always be eliminated. If b is an (n X n) plane section of the array B then an r-partial solution of b is a set of r independent true elements, ie. r true elements such that no two occur in the same row or column. An n-partial solution is the time-table for the class, teacher or hour represented by b.

The examination of b has two stages:

1/ confirmation of the existence of at least one schedule (feasibility test)

2/ any true element which does not belong to a possible schedule is changed to false (matrix reduction)

The feasibility test is a very simple procedure but the matrix reduction is a highly complex process requiring large amounts of computing time. The proposed "tight set search" procedure for reducing the matrix can be shown to converge to a solution in about 2^n steps and thus is, unfortunately, impractical if $n \geq 20$. J. Lions (24) has developed some refinements to the tight set search which will reduce the effort required to about $(mn)^2$ operations, where m is the number of <u>true</u> elements in b.

The largest reported time-table produced by this method is one for a school of 9 rooms, 9 teachers, and 9 teaching hours. Lions has used the method to produce master time-tables for schools in Ontario; however, to the author's knowledge, none of his time-tables have ever been openly published.

The last method under consideration is one developed, on an experimental basis, by J. Pfaltz (29) of the University of Maryland. His heuristic procedure is very crude because he did not carry his work to completion, however it shows promise of becoming a very useful system.

Pfaltz conceived of the procedure being used after the student body had enrolled, thus it would have an accurate record of the number of students enrolled in each course and the number of students taking any pair of courses. The registrar was expected to supply, as input data, the following information:

1/ Course data

- name
- length of class eg. 1 hour 3 times a week
- two preferred times for class meeting, or to be arranged, if necessary
- the maximum number of students to be allowed in any one section of the course

2/ Available time periods

- a list of all hours, suitable for teaching in any one week

3/ The aforementioned registration data.

The procedure will perform the following basic steps in its attempt to design a time-table:

1/ Tabulate all the registrations for each course and form a list of the courses conflicting with each course.

2/ Compare the total registration with the maximum number of students permited in each section, and decide how many sections of each course are to be offered.

3/ Form a priority list of the classes to be scheduled. This ensures that classes which are difficult to schedule are attempted first. The priority list is based on the total enrolment, whether it is a single or multiple sectioned course, and the length of time a class is to meet over one week.

4/ Modify ~~es~~ the conflict lists of the multiple sectioned courses so that only the most serious possible conflicts remain. The basic philosophy behind this is quite simple - if two single section courses have students common to both, then they cannot meet at the same or overlapping times, however if one of the courses has been divided into two sections there is some possibility that assigning students to the other section will resolve the apparent schedule conflict. If one, or both, of the conflicting courses has more than three sections then Pfaltz ignores the apparent conflict.

5/ The final step is to assign the top unscheduled course in the priority list to the time period of its choice, if possible, otherwise to any free time period. The routine then attempts to assign as many courses as possible to the same time period; however if any of the first five courses, in the non-conflicting list, prefers that time period it gets it, even though it is out of strict priority. This added queue jumping does not seem to adversely affect the system and will go a long way in making it more palatable to the user.

In the limited tests Pfaltz made, the procedure seemed to work very well. There are, however, a number of improvements which could be made:

1/ Pfaltz arbitrarily chose some of the parameters (such as the number of conflicts which the procedure could safely ignore). Further study could refine these and perhaps suggest others.

2/ As the program tries to schedule all the courses in as few time periods as possible, the resulting distribution of courses over the week is very uneven. Some form of levelling routine will be necessary, this may even be accomplished by modifing the values of the parameters mentioned in point 1 above.

Of all the master time-table procedures available in the literature none is entirely satisfactory. The most fruitful avenue of approach is, perhaps, to combine the clerical reducing ideas of Holtz with the easily modified heuristics of Pfaltz to produce a system which, if not perfect, would be extremely useful.

Section 3.3 A New Approach

The problems involved in producing examination time-tables and the sectioning of students to classes are essentially one and two dimensional assignment problems. All of the structures dealt with in Chapters 1 and 2 involved only relations between pairs of items, often called dyadic relations. The master time-table of an institution, on the other hand, requires one to deal in tetradic relationships, ie. student A meets with teacher B in room C at time D. This may be simplified to a triadic relation if the teacher is considered as simply another student required to attend the class.

A fruitful mathematical theory for n-adic relations ($n > 2$) seems to be undiscovered. This is essentially due to the fact that dyadic relations correspond to matrices and standard matrix operations have a definite meaning in terms of dyadic relations. However n-adic relations correspond to n-dimensional matrices and the handling of these matrices presents a number of special problems.

To be able to apply the techniques developed in Chapters 1 and 2 to the problem of producing a master time-table some method must be developed to reduce the problem to simple dyadic relations. In the past, the attempts at building a master time-table have considered rooms, students, teachers, and times as separate entities and thus were forced into three or four dimensional assignment problems.

In some scheduling problems, notably problems arising out of shop floor and assembly line scheduling, not all assignments have to be made at once, for example a piece of work may not have a machine or operator available for it, and thus it can simply be put into a waiting queue. On the other hand, a class with a room but no teacher is quite a useless assignment. Thus students, teachers, rooms, times and any special equipment required are all of equal importance in the consideration of master time-tables.

In order to avoid confusion in the discussion on master time-table preparation the following definitions are necessary:

- a "primitive facility" (or "primitive") will denote a particular teacher, room, time segment, piece of equipment, group of students, etc.
- a "resource" will denote a collection of identical (or interchangeable) primitives, eg. a number of teachers having the same qualifications, a number of rooms of equal capacity etc..
- a "class" will denote a collection of primitives (one from each necessary resource), a class will normally be given a name, eg. the class called

jr. mathematics may consist of the primitives:

- Mr. White (the teacher)
- room 709
- the first year group of mathematics students
- the time segment Mon. Wed. Fri. 9:00-10:00
- the first year mathematics demonstration kit.

- a "time-table" will denote the collection of all the classes.

Consider a student enrolling for studies. In Chapter 2 it was seen how he could produce a list of the courses he wished to attend and how a computer could assign him specific sections of each course. If, in place of this student, we substitute a department head, he could produce a list of resources necessary for the formation of a class. For example, if he was preparing a list of resources for jr. chemistry it might consist of:

1/ a teacher of chemistry

2/ a lecture room holding 200 students, with a demonstration bench

3/ the first year class of science students

4/ a chart of the periodic table of elements

5/ a time segment consisting of three single hours per week.

Each of these resources is composed of one or more primitives, for example:

1/ (teacher) a four section resource

a) Mr. White

b) Mr. Brown

c) Mr. Green

d) Dr. Red

2/ (room) a two section resource

a) room 707

b) room 103

3/ (pupils) a single section resource

4/ (chart) a three section resource

a) chart A

b) chart B

c) chart C

5/ (time) a ten section resource

a) Mon. Wed. Fri. 8:00 - 9:00

b) Mon. Wed. Fri. 9:00 - 10:00

c) Tues. Thurs. Sat. 8:00 - 9:00

etc..

The process of choosing one primitive from each resource to form a class is easily seen to be identical to the problem of sectioning students to classes.

The full power of an algorithmic sectioning procedure is vital to the successful production of a workable time-table by this method. Each primitive may be represented in the same manner as actual course sections were in Chapter 2, the section vector and reservation number concepts acting as powerful selection criterias for obtaining the correct assignment of teachers, rooms, times, equipment, etc..

The boolean matrix, T, representing the master time-table in Chapter 2, is now replaced by a boolean matrix P having one row and column for each primitive available. Initially P_{ij} = false indicating that any pair of primitives may be assigned together. As the assignments are made for each class the matrix P is updated to show the conflicts of the primitives in the "time" resource with the primitives in the other resources.

In practice the matrix P will not be easily set up. For example the "time" primitives may have to be modified to avoid overlapping time periods, and a great deal of study will be needed to accurately determine all the primitives within an institution. Although unwieldy to set up, the matrix may be used to great advantage, for example, if a piece of equipment, i, is available only in the chemistry building then by assigning

$$P_{ij} = \underline{\text{true}} \text{ (j = all rooms not in chemistry building)}$$

this will ensure that a class requiring equipment i will be held in the chemistry building.

Unlike student sectioning, where sections of one course formed a completely disjoint set with sections fro..

another course, the primitives in one resource may be identical with some or all of the primitives in another resource, eg. a teacher qualified in both chemistry and mathematics. Thus great care must be taken in ensuring that there is one and only one row and column in P for each primitive. This also forces a slight change in the sectioning algorithm. Step 4, the generation of edges between vertices of the individual sections (primitives) of one course (resource) must now be accomplished by setting P_{ij} = true if primitive i and primitive j are in the same resourse, for each resource in which primitive i and j are grouped.

To ensure an even distribution of the workload to each primitive in a resource it is only necessary to keep a record (corresponding to the number of students in each section) of the number of hours each primitive is occupied. The sectioning procedure, by means of the section vector, will attempt to assign the primitive with the least usage before attempting to use primitives in greater demand.

It would be possible to use this method by selecting at random a department head's list of requirements, sectioning them, and selecting another list of requirements This, however, will rapidly lead to a situation in which it is impossible to find a valid assignment of primitives for one or more classes. This is equivalent to the random selection of vertices for colouring, a process seen, in

Chapter 1, to produce far from optimum results.

The selection order of the lists of requirements may be obtained by the same methods as the selection order for vertices in the graph colouring problem. An N X N matrix R (where the institute wishes to offer N classes) is produced such that

$$R_{ij} = s$$

if s requirements of class i are the same as those of class j. The eigenvector is found corresponding to the largest eigenvalue of the matrix R (or an approximation to this eigenvector - see Section 1.5) then the classes for assignment of primitives are chosen in the order of decreasing magnitude of the elements of this eigenvector (see Section 1.3).

This ordering criterion may be changed in individual situations by having an extra weighting factor of two or three on the conflict of room requirements if the institute is short of space, or teachers if it is short of staff, etc.. This will tend to raise the magnitude of the eigenvector elements corresponding to the weighted requirements.

Because the various sections of a course will normally have the same primitives, they will have equal values of their elements in the eigenvector and thus be sectioned one after another. This together with the distributing function of the section vector will ensure that they are sectioned in different time primitives. A conflict list of the type used by Pfaltz (note point 3 in the description of Pfaltz's procedure in Section 3.2) may also be useful in ensuring an even distribution of assignments.

If an extremely powerful computer is available an alternate approach will produce "best possible" results. The graph, G_i, corresponding to the non-conflicts between primitives for any one particular requirement, i, (the graph from which a complete n graph is found, giving a workable sectioning) will normally contain more than one complete n graph, these complete n graphs will be denoted by γ_i^1, γ_i^2, γ_i^3, ...γ_i^k.

The following procedure will determine all the possibl time-tables for an institution:

1/ Produce all the γ_i for each requirement i

2/ Form a graph, Γ, the vertices of which correspond to the γ_i produced in step 1

3/ If there exists γ_i^1, γ_i^2, γ_i^3, ...,γ_i^n for any requirement i then connect the vertices corresponding to the n γ_i such that they form a complete n graph

4/ Take the time primitive associated with a vertex γ_i^k and compare it ~~to~~ with the time primitives associated with each vertex γ_j^l ($i \neq j$). If the two time primitives conflict and a student has requested both course i and course j then join γ_i^k by an edge to γ_j^l.

5/ Take the complement of Γ. $\overline{\Gamma}$ is now the graph of all the γ_i which are pairwise compatible

6/ If the administration specified N classes then from $\overline{\Gamma}$ find all the complete graphs of order N. These represent all the possible time-tables the institution may use.

Once again the repeated use of the complete graph sectioning algorithm will yield the solution to a very difficult problem.

The complete graph method, suggested in the last section, for the construction of school time-tables was put forward as a serious suggestion. However, in practice, a number of drawbacks make themselves felt when it comes to designing a practical implementation of the system. A practical system would require a large amount of data in a rather unnatural form and, if the data were collected, it would require a very large and powerful computer to analyse it.

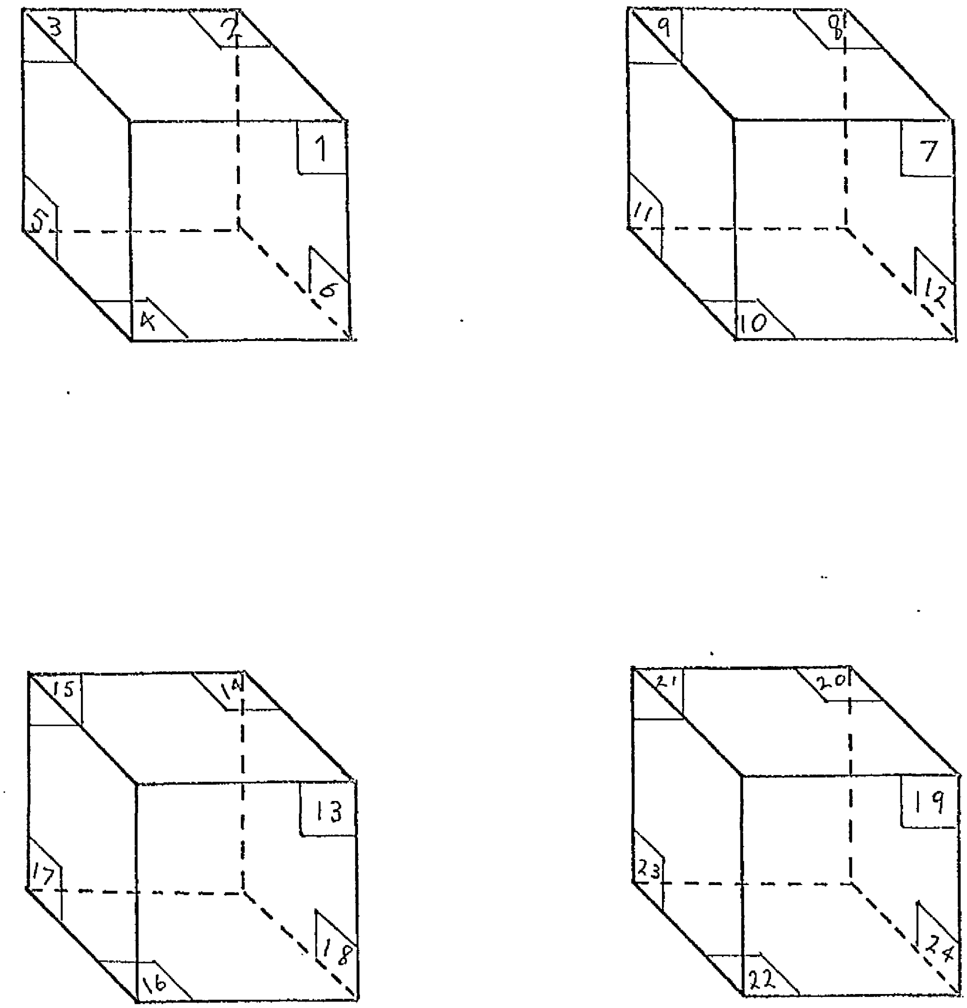
To collect and classify all the resources of a university would be an extremely arduous task. In fact it is likely that the total resources or physical facilities available to a university are never exactly known. In 1964 the Office of the Registrar of the University of Alberta, Calgary attempted to classify all the lecture, laboratory, and seminar rooms and their contents; to the Author's knowledge this survey was never completed and finally abandoned completely due to its complex nature.

The Author has had the privilege of attending the I.F.I.P. Congress 68 (Edinburgh 1968). During the Congress H. C. Johnston and K. Wolfenden presented a paper entitled "Computer Aided Construction of School Time-tables" which described a method of collecting school data in a form similar to that required by the complete graph method. Discussion after the presentation indicated that this data collection, for a small school, was a nontrivial and error prone task.

From the experience gained in running the complete algorithm for student sectioning (and a project described in Section 4.3), it was forecast that a realistic attempt at constructing a master time-table would take of the order of tens of hours of KDF 9 computer time. However (again from experience gained on implementing the sectioning algorithm) it should be possible to increase the efficiency of the algorithm from 150% to 300% by very careful hand coding. This would not only be a huge job, it would mean hand coding of recursive routines, a messy business at the best of times.

Because of these difficulties, and a lack of both the time and the money to find methods of overcoming them, it was decided that an implementation of the system was not a practical possibility, especially on a computer the size of a KDF 9. It was possible, nevertheless, to solve a small problem with this master time-table procedure and, in so doing, learn more about its operation.

In the spring of 1968 a toy, claiming to be an invention of one Dokter Adlor, called Instant Insanity came on the market. The pieces of this game are four cubes with the faces coloured red, green, blue, and



red 1,7,8,9,15,21,23

yellow 2,6,12,16,19,20

green 3,5,10,17,18,22

blue 4,11,13,14,24

FIGURE 3.4.1

Showing the colour scheme of Doktor Adlor's cubes.

yellow. The object is to place the cubes in a line to form a solid rectangle such that each colour is represented once and only once on each face of the solid rectangle. The cube colour scheme is that shown in FIGURE 3.4.1.

Although simple in concept it is extremely difficult to find a solution. The author has never been able to find a solution and knows of only a few people who, generally after a month or more, have discovered one. However it is possible to formulate this problem in terms of graph theory and, because of its resemblance to the school time-table problem, construct an algorithm to tabulate all possible solutions.

Form a graph, g, of 24 vertices representing the 24 faces of the four cubes; if face i is on a different cube from face j and has a different colour from face j then vertices i and j are joined by an edge. The graph g now represents those faces which may be placed adjacent to each other to form one half of a face of the solid rectangle. If all of the complete graphs of order four are found then these K_4's will represent an arrangement of one face from each cube such that when placed together they form a solution to one face of the solid rectangle. It should be noted that this one face solution implies nothing about the other three faces in any possible solution.

The graph g was formed and submitted to the complete graph procedure which produced 130 one face solutions, or 130 ways of arranging the cubes such that at least one face of the solid rectangle shows all four colours. Because of the symmetry of the situation many of the 130 K_4's from g may be eliminated from consideration in the final solution. Specifically the K_4's corresponding to a single face solution may be eliminated if the face on the opposite side of the solid rectangle is not also a one face solution.

After eliminating the useless one face solutions a graph, G, may be formed, each vertex of which corresponds to one of the remaining K_4's of g. Vertex i is joined to vertex j in G if the two one face solutions corresponding to i and j are compatible. This compatibilit is achieved if:

1/ no face of a cube used in solution i is used in solution j

2/ if the face of one cube used in solution i is on the opposite side of the cube to the face used in solution j then the three other faces must also be opposite in i and j.

Any complete graph of order four in G will now represent four compatible one face solutions or the final answer to Doktor Adlor's problem.

The elimination of useless K_4's from g leaves six of the K_4's as being possibly contained in a solution namely:

1/ 1,12,13,22

2/ 4,10,15,20

3/ 6,10,14,23

4/ 3,11,15,20

5/ 2,8,13,22

6/ 5,8,16,24

These give rise to the graph G shown in FIGURE 3.4.2. It is easily seen that there is only one K_4 in G and thus only one solution to Doktor Adlor's problem, namely:

face 1 - 1,12,13,22

face 2 - 6,10,14,23

face 3 - 3,11,15,20

face 4 - 5,8,16,24.

This rather trivial problem serves as an example of the use of one graph representing groups of vertices and their relations in another graph. It thus gives an elementary problem to examine which will help determine the problems involved in implementing this type of approach. In general it can be said that a problem of this size is easily dealt with on even small computers although a small increase in its complexity

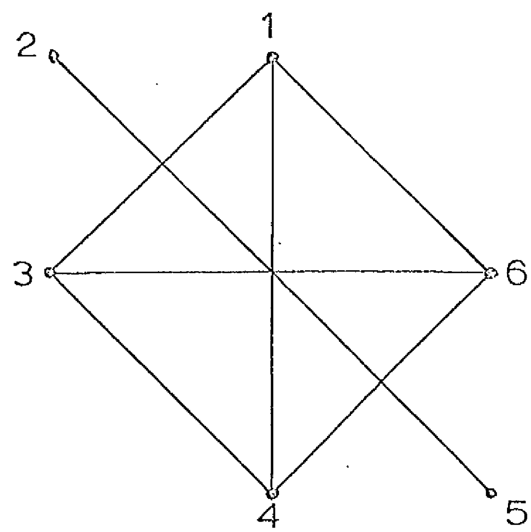


FIGURE 3.4.2

The graph of the one face solutions to Doktor Adlor's problem.

could result in the computer expending many times the effort to obtain a solution.

The main difficulty would seem to be in the design of procedures to handle the data between uses of the complete graph procedure. In particular the design of efficient procedures to eliminate the useless or redundant partial solutions is the task which could make or break this approach.

CHAPTER 4

Further Results

Section 4.1 A Bound for the Chromatic Number

In this chapter an investigation is made of the graph colouring processes and related topics. A few interesting results are obtained and several valuable insights into computational processes are found. Several theorems will be valuable for the future discussion.

Theorem 4.1.1

If a graph, $G(V,U)$, without loops or parallel edges has an associated matrix A, then two vertices i and j may be given the same colour if row i of A is identical to row j of A.

Proof

It is known that vertices i and j may be given the same colour if they are not connected by an edge and if they are not both in a circuit containing an odd number of edges. Thus the proof can be split into two parts.

1/ Assume that vertex i and vertex j are joined by an edge, then

$$a_{ij} = 1 \quad \text{and} \quad a_{ji} = 1$$

for row i to be identical to row j this would force

$$a_{ii} = 1 \quad \text{and} \quad a_{jj} = 1$$

which contradicts the assumption that G has no loops. Therefore vertex i is not joined to vertex j.

2/For row i to be identical to row j there must exist a set of vertices K such that

$$a_{ik} = 1 \quad \text{and} \quad a_{jk} = 1$$

which implies that there must exist a circuit from i to j and back to i of the form

$$i,\ k,\ j,\ k,\ i$$

where k is any member of the set K. This circuit has an even number of edges and obviously is the only type of circuit in existence.

This proves the theorem.

This result may be used to eliminate from the graph any vertices which, by reason of having an identical twin, are definitely not part of a critical subgraph.

A stronger result is:

<u>Theorem 4.1.2</u>

If row j in the matrix A, associated with the graph G(V,U) (no loops or parallel edges), is a linear combination of the rows

$$i_1, i_2, \ldots\ldots, i_p$$

then the vertices corresponding to the rows

$$i_1, i_2, \ldots\ldots, i_p$$

may be given the same colour as the vertex corresponding to the row j.

Proof

The vertices $i_1, i_2, \ldots\ldots, i_p$ must be at a distance two from vertex j and only j. There must exist p nonempty sets $K_1, K_2, \ldots, K_p$ such that

$$i_r \in U_{K_r} \quad \text{and} \quad j \in U_{K_r} \qquad (r = 1,2,..p$$

and

$$i_r \notin U_{K_s} \qquad (\text{for all } r \neq s)$$

It will only be necessary to show that any of the vertices i_r may be given the same colour as j and that this colouring of i_r does not effect the colours assigned to the other p - 1 vertices.

Assume that the graph has been coloured in a minimal number of colours and that vertex j was given the colour α. If the vertex i_r has also been given the colour α then there is no problem, however if it has been given the colour β then consider the following -

No vertex in K_r may be coloured α because

$$j \in U_{K_r}$$

or β because

$$i \in U_{K_r}$$

By an argument similar to that in theorem 4.1.1 concerning the possible joining of vertex i_r and j and the length of the smallest circuit containing both vertices it is possible to show that vertex i_r may be given either the colour α or the colour β, thus one may change the colour of i_r to α.

The colouring of i_r must be independent of the colourings of the other p - 1 vertices!

The relation

$$K_r \wedge K_m = \emptyset$$

must hold true otherwise i_r and i_m would be at a distance two from one another, and the vertex k satisfying the conditions

$$k \in U_{i_r} \qquad k \in U_j \qquad k \in U_{i_m}$$

would force the element

$$a_{jk} > 1$$

which violates the condition that G contain no parallel edges. Thus i_r and i_m are at a distance four from each other and thus do not have to be given different colours by reason of being on an edge circuit of odd length.
Finally

$$i_r \notin U_{i_m}$$

because the reverse implies that i_r is included in one of the sets K_m. If

$$i_r \in U_{i_m}$$

then

$$i_r \in U_j$$

which has already been show· to be impossible.
Thus the theorem is proved.

Theorem 4.1.2 may be used in any attempt to locate the critical chromatic subgraph of a large graph. This result leads directly onto another giving an upper bound for the chromatic number of a graph.

<u>Theorem 4.1.3</u>

The chromatic number of a graph, $\gamma(G)$, obeys the relation

$$\gamma(G) \leq R$$

where R is the rank of the matrix A associated with the graph G.

<u>Proof</u>

The rank of a matrix is the number of linearly independent rows and columns of the matrix. Theorem 4.1.2 allows the deletion of all dependent rows and columns without changing $\gamma(G)$, therefore

$$\gamma(G) \leq R.$$

Equality holds in theorem 4.1.3 for complete graphs on n vertices ($R = n$). Thus this upper bound is the best possible for general graphs.

Recently Szekeres and Wilf (44) have published (without proof) a potentially better bound for $\gamma(G)$. Their bound is:

$$\gamma(G) \leq \lambda_1 + 1 \qquad (4.1.4)$$

where λ_1 is the largest eigenvalue of the matrix associated with the graph G. Results deduced in Section 4.3 will throw doubt on the validity of (4.1.4) for general graphs.

Section 4.2 Graphical Reduction

The largest factor in determining the efficiency of a graph colouring procedure is the size of the graph itself. If some method could be found to easily reduce the size then the computation necessary to produce a minimal colouring will be greatly reduced.

A very much weaker (but computationally more significant) result than Theorem 4.1.2 is the following:

Theorem 4.2.1

> If two rows of a matrix A (associated with a graph G having no loops or parallel edges) bear the relation that the nonzero elements of row i are a subset of the nonzero elements of row j then the vertex i may be given the same colour as the vertex j.

Proof

> The proof follows the same general lines as the proof of theorem 4.1.1.

This theorem is computationally important because it is very easily programmed on most computers in such a manner that the full power of the computer's logical instruction set can be brought to bear on the problem of determining if row i is a subset of row j. This particular operation (that of reducing the number of vertices in a graph by eliminating subset rows) will be known as reducing the graph. A graph G, having been fully reduced, will be denoted by G^r, similarly its associated matrix will be denoted by A^r.

If only the chromatic number is wanted then an alternate method of reduction is available involving the cartesian product of two graphs. This method is only useful on a limited number of graphs but if the physical problem being represented has some symmetry to it then this method may be applied.

The cartesian product of two graphs $G(v,u)$ and $H(z,w)$ (both having no loops or parallel edges) is the graph $\Gamma(V,U)$ whose vertices are the ordered pairs (x,y) where $x \in v$ and $y \in z$ and (x,y) is adjacent to (x',y') if and only is

1/ $x = x'$ and y is adjacent to y' in H

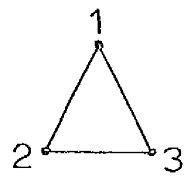
or

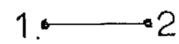
2/ $y = y'$ and x is adjacent to x' in G

V. G. Vissing (34) has shown that

$$\gamma(\Gamma) = \max\left\{\gamma(G),\ \gamma(H)\right\}$$

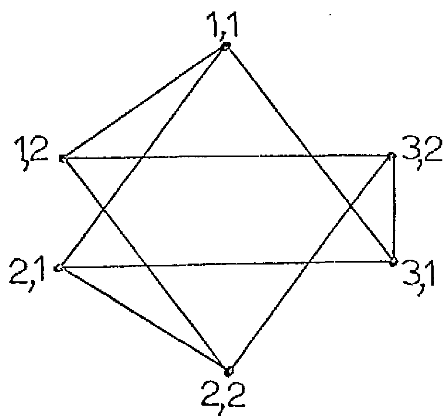
thus if it is possible to factor the original graph into two cartesian factors G and H the work needed to

G = 

H = 

	1	2	3
1	0	1	1
2	1	0	1
3	1	1	0

	1	2
1	0	1
2	1	0

Γ = 

	1,1	1,2	2,1	2,2	3,1	3,2
1,1	0	1	1	0	1	0
1,2	1	0	0	1	0	0
2,1	1	0	0	1	1	0
2,2	0	1	1	0	0	1
3,1	1	0	1	0	0	1
3,2	0	0	0	1	1	0

FIGURE 4.2.1

Showing two graphs G and H and their cartesian product Γ.

find the chromatic number may be reduced by several orders of magnitude.

To factor Γ into G and H it is obvious that

$$|v| \times |z| = |V|$$

If $|z| < |v|$ then the δ_i of Γ must have $|z|$ symmetry, ie. if $|z| = 3$ then $|V| \div 3$ must be integral and there must be at least 3 δ_i's of Γ with the same value. ~~for all δ_i in Γ~~ Inspection of FIGURE 4.2.1 will clarify the matter.

A simple test to determine if it is possible that G and H are the cartesian factors of Γ is to determine that the following relation always holds true for the degree (δ) of each vertex in Γ.

$$\delta_i \leq |v| + |z| - 2$$

If this is not the case then G and H can not possibly be cartesian factors of Γ.

Graphs with this type of symmetry are not common. Because of this, and the effort necessary to deduce G and H, cartesian factoring is only usefully employed in cases where this type of symmetry is known to exist.

Section 4.3 The Eigenvalues and Eigenvectors of a Graph

The use of an eigenvector as the ordering criterion of a colouring procedure naturally led to the problem of the meaning, in graphical terms, of all the eigenvalues and eigenvectors of the matrix associated with a graph. Considering, for a moment, only undirected graphs with no parallel edges (ie. the associated matrix is symmetric, the elements are either 0 or 1, and there are 1's down the leading diagonal) it is evident that this array may be considered as a correlation matrix, a pair of vertices joined by an edge having a correlation coefficient of 1 and other pairs of vertices having a correlation coefficient of 0. Taking this view of a graph it is possible to find explanations for the eigenvalues and eigenvectors of a graph in the body of knowledge built up around that part of multivariate statistical methods known as principal component analysis.

Investigators in the behavioural sciences are often faced with the problem of having data of a series of observations on several aspects of one individual, or the correlations of these observations on several individuals. As these observations are all drawn on a single individual there will clearly be some dependence relationship between them. Principal component analysis is one of the methods of elucidating this dependence structure. In general this dependence will be based on a number of factors, each of which will add its own component to the system's structure.

Morrison (26) shows that the Jth principal component of a system is a linear compound

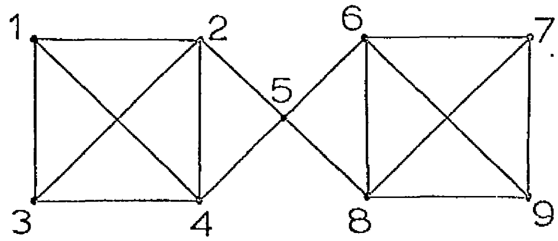
$$Y_j = a_1X_1 + a_2X_2 + \ldots\ldots + a_pX_p$$

of the observations, X_i, whose coefficients, a_i, are the elements of the eigenvector of the correlation matrix A corresponding to the Jth largest eigenvalue, λ_j. The importance of the Jth component, I_j, in describing the dependence structure is:

$$I_j = \frac{\lambda_j}{tr(A)} \qquad (4.3.1)$$

where tr(A) is the trace of the matrix A.

The graphical interpretation of this is evident from FIGURE 4.3.1 in which a graph of nine vertices and sixteen edges is displayed along with four of its eigenvalues and eigenvectors. The other five eigenvalues are all equal to zero and thus by EQUATION 4.3.1 have no significance in describing the structure. The first eigenvector, as was seen previously, gives a measure of how deeply embedded a vertex is, or a measure of its ability to dominate the other vertices in the graph. As expected, vertex 5 comes out as the most dominant, followed by vertices 2, 4, 6, and 8 which in turn are



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

eigenvalue = 4.6262
corresponding vector
0.27740
0.36426
0.27740
0.36426
0.40181
0.36426
0.27740
0.36426
0.27740

eigenvalue = 4.0000
corresponding vector
0.35355
0.35355
0.35355
0.35355
0.00000
-0.35355
-0.35355
-0.35355
-0.35355

eigenvalue = 1.5151
corresponding vector
0.35855
-0.08692
0.35855
-0.08692
-0.67495
-0.08692
0.35855
-0.08692
0.35855

eigenvalue = -1.1413
corresponding vector
-0.21093
0.33130
-0.21093
0.33130
-0.61887
0.33130
-0.21093
0.33130
-0.21093

FIGURE 4.3.1

Showing a graph, its associated matrix, and its four unique eigenvalues and eigenvectors.

closely followed by 1, 3, 7, and 9. EQUATION 4.3.1 shows that the first eigenvector explains 52.5% of the dependence relation in this structure. The second eigenvector, explaining 44.5%, points out the locally compact equal groups of vertices 1, 2, 3, 4 and 6, 7, 8, 9. The third eigenvector, explaining 16.7% of the structure, presents what amounts to two divisions of power or dominance, one the outer and central vertices 1, 3, 5, 7, and 9, and the other the inner structure 2, 4, 6, and 8. The final eigenvector corresponds to a negative eigenvalue, indicating that the previous three eigenvectors have over-specified the structure by 13.7% and this vector will help correct the situation.

Consider an n X n matrix of the form:

$$\begin{matrix} 1 & p & p & \cdots & & p & p \\ p & 1 & p & \cdots & & p & p \\ p & p & 1 & \cdots & & p & p \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ p & p & p & \cdots & & 1 & p \\ p & p & p & \cdots & & p & 1 \end{matrix}$$

if

$$0 \leq p \leq 1$$

then the largest eigenvalue of this matrix, λ_1, is

$$\lambda_1 = 1 + (n - 1)p$$

and the corresponding eigenvector, u_1, is

$$u_1 = \left[\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}} \cdots\cdots \frac{1}{\sqrt{n}}\right]$$

which may be scaled up to

$$u_1 = [1,1,1,\ldots,1]$$

The other n-1 eigenvalues are

$$\lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = \;\ldots\ldots\; = \lambda_n = 1-p$$

and the corresponding eigenvectors are all orthogonal to u_1. Thus a complete graph on n vertices has an associated matrix, all of whose elements are 1, with

$$\lambda_1 = 1+n-1 = n$$

$$\lambda_2 = \lambda_3 = \lambda_4 = \;\ldots\; = \lambda_n = 0$$

Consider a positive definite n X n matrix A (all of whose elements are either 0 or 1) of rank r (r = n). A can be expressed as

$$A = \sum_i \lambda_i u_i^t u_i$$

if each

$$u_i\sqrt{\lambda_i} = 1 \quad (\text{for } i = 1,2,3,\ldots,n)$$

then all the eigenvectors are of the form

$$u_i = [u_{i_1}, u_{i_2}, u_{i_3}, \ldots, u_{i_n}]$$

with

$$u_{i_k} = \text{either 0 or 1}$$

Because all the eigenvectors must be orthogonal it is clear that if

$$u_{i_j} = 1$$

then

$$u_{k_j} = 0 \quad (\text{for all } k \neq i)$$

As an example consider the matrices A_1 and A_2 in FIGURE 4.3.2. The matrices A_1 and A_2 correspond to the graphs G_1 and G_2 in FIGURE 4.3.2.

The complement of a graph G, denoted by $\overline{G}$, is the grap formed from the same vertex set as G but having edges according to the following rule:

if vertex i and vertex j are joined in

G then they are not joined in $\overline{G}$ and if vertex i and vertex j are not joined in G then they are in $\overline{G}$.

An examination of the graphs $\overline{G_1}$ and G_1 and their associated matrices $\overline{A_1}$ and A_1 shows that a minimal colouring of the vertices of $\overline{G}$ may be obtained by the following rule

vertex i is given colour j if and only if $u_{j_i} = 1$.

Thus the eigenvectors of the matrices A_1 and A_2 represent colour groups of the vertices of $\overline{G_1}$ and $\overline{G_2}$.

Given a graph G and its complement $\overline{G}$ it is obvious that the matrix $\overline{A}$, associated with $\overline{G}$, will not always have eigenvectors of the form

$$\sum_i \overline{u}_i\sqrt{\overline{\lambda_i}} = 1. \qquad (4.3.2)$$

However the following theorem eases this particular difficulty.

<u>Theorem 4.3.1</u>

If a graph G, of chromatic number $\gamma(G)$, has extra edges added to it to form the graph G' then the chromatic number of G', $\gamma(G')$, obeys

$$A_1 = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 & 1 \\ 3 & 1 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 0 & 1 & 0 & 1 \\ 5 & 1 & 0 & 1 & 0 & 1 & 0 \\ 6 & 0 & 1 & 0 & 1 & 0 & 1 \end{array} \qquad A_2 = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 1 \end{array}$$

$\lambda_1 = 3 \quad u_1 = [101010]$

$\lambda_2 = 3 \quad u_2 = [010101]$

$\lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 = 0$

$\lambda_1 = 1 \quad u_1 = [10000]$

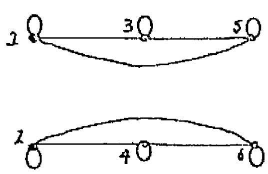
$\lambda_2 = 1 \quad u_2 = [01000]$

$\lambda_3 = 1 \quad u_3 = [00100]$

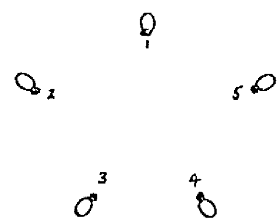
$\lambda_4 = 1 \quad u_4 = [00010]$

$\lambda_5 = 1 \quad u_5 = [00001]$

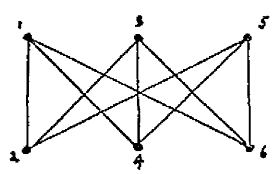
$G_1 =$



$G_2 =$



$\overline{G}_1 =$



$\overline{G}_2 =$

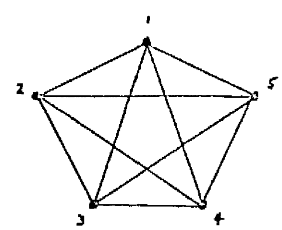


FIGURE 4.3.2

Showing two matrices, their eigenvalues and eigenvectors, and their corresponding graphs and converse graphs.

the relation

$$\gamma(G) \leq \gamma(G').$$

Proof

Obvious.

If edges are added to G then the corresponding edges are deleted from $\overline{G}$. Thus it should be possible to define a matrix B and consequently a matrix C such that

$$\overline{A} - B = C \qquad (4.3.3)$$

and such that condition (4.3.2) holds for the matrix C. An examination of the properties of the matrix B will shed some light on graph colouring processes.

An internally stable set of vertices is a set such that no two members of the set are adjacent. The coefficient of internal stability is the number of vertices in the largest internally stable set and is denoted by $\alpha(G)$.

Because of the relation (4.3.2) there must be exactly $\overline{\lambda}_i$ vertices with the colour i in any graph corresponding to the matrix C in (4.3.3). That is to say

$$\overline{\lambda}_i = |v|$$

where v is an internally stable set of vertices from G.

It is known that

$$\alpha(G)\,\gamma(G) \geq n \qquad (4.3.4)$$

where n is the number of vertices in the graph. It is not possible, however, to colour a graph by finding the largest internally stable set and giving this colour 1, then finding the next largest internally stable set etc.. This is easily demonstrated by the graph in FIGURE 4.3.3. The largest internally stable set are those vertices represented by circles. If these are given colour 1 then the remaining three vertices must be given colours 2, 3, and 4. Four colours have then been used when the chromatic number of the graph is only three. If the graph is reduced, using Theorem 4.2.1 then this difficulty is overcome in G^r.

For a reduced graph the coefficient of internal stability is equal to the number of vertices having the most popular colour and, by the previous argument

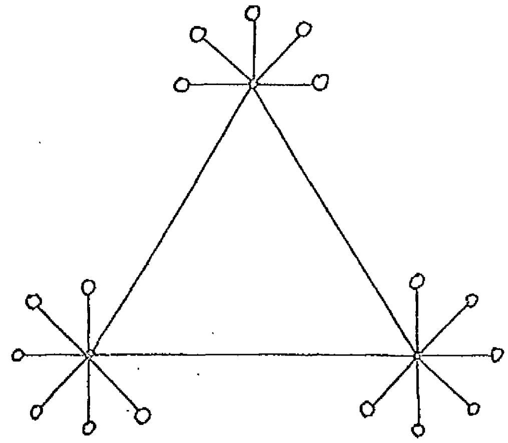
$$\alpha(G) \leq \overline{\chi_1^r}$$

Thus relation (4.3.3) may be rewritten as

$$\overline{\chi_1}\,\gamma(G) \geq n$$

and

$G \quad =$



$G^r =$

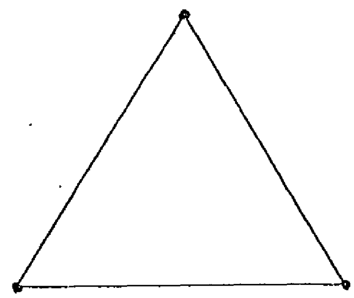


FIGURE 4.3.3

Showing a graph G and its reduction G^r.

$$\gamma(G) \geq \frac{n}{\chi_1}$$

Normally this is not a very good lower bound, however it may be used as a first approximation.

After establishing these few basic relations it is possible to return to the problem of finding the matrix B in equation (4.3.2). Several properties of B are immediately obvious. If B is the null matrix then

$$\bar{A} = C$$

and $\bar{A}$ is in the form of a block diagonal matrix, or may be put in the form of a block diagonal matrix by a suitable interchange of rows and columns (ie. by renumbering the vertices). If $\bar{A}$ is of block diagonal form then G is said to be k-partite, where k is the number of blocks along the diagonal.

If it is possible to find a matrix B then the matrix C is very likely to have decreased in rank with respect to $\bar{A}$. In fact by making

$$B = \bar{A}$$

it is possible to cause C to be the null matrix and thus arrive at the degenerate condition of one colour for each vertex. Thus it is necessary to specify that B contain the minimum number of elements equal to 1 and that the rank of B also be kept to a minimum.

It is possible to set up the solution of B as a linear programming problem. However this involves a double minimization of the rank of B and the number of elements of B which leads to the same consequences as the original linear programming formulation of the colouring problem in Section 1.2 (which, considering it is really the same problem, is not suprising).

An alternate algorithmic approach to finding B is contained in the simple statement

$$K(\overline{G^r}) = \alpha(G^r)$$

where K(G) is the size of the largest complete subgraph in the graph G.

Each block along the diagonal of C represents the vertices given each colour. The largest complete subgraph of $\overline{G}$ is the largest block along the diagonal of C. Thus the complete graph algorithm may be used repeatedly to determine each block of C and thus indirectly determine B.

This is an algorithmic method of colouring a graph which possesses none of the disadvantages of the other algorithms and the advantage that, once the complete graph algorithm has been implemented, is easily programmed.

A byproduct of this investigation is the fact that

$$\alpha(G) \leq \overline{\lambda},$$

and

$$K(G) \leq \lambda_1$$

The bound by Szekers and Wilf mentioned in Section 4.1 (4.1.4) is not, of necessity, valid because the relation

$$K(G) \leq \gamma(G) \leq K(G) + 1$$

is not always true and thus

$$\gamma(G) \leq \lambda_1 + 1$$

is not valid for all graphs. Even though this may not be valid it is still suitable as a first approximation. The accuracy of this approximation may be judged from the fact that the U.A.C. 1965 - 66 data is colourable in 28 colours (using the heuristics developed in Chapter 1) while its largest eigenvalue was 79.4.

There may or may not be a unique matrix B and thus a unique colouring of G^r. There is likely to be at least one vertex which may be given either colour i or colour j. If one colour group is too large then the following procedure will redistribute the colours of G in a more even fashion. Take a colour group I consisting of the p+1 vertices $i_r, i_{r+1}, \ldots, i_{r+p}$.

If there exists a row j of the matrix B such that

$$B_{i\,j_q} = 1 \text{ (for } q = r, r+1, r+2, \ldots, r+p)$$

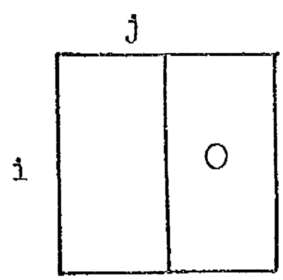
then vertex j may be added to colour group I.

It should be possible to check the heuristic calculations of Chapter 1 by this complete graph method. Unfortunately the amount of computation necessary increases greatly as the number of edges in the graph increases. The converse of both data sets used in Chapter 1 contained so many edges that over six hours of KDF 9 time failed to yield a solution. However subgraphs from these data sets, consisting of 50 vertices each, were run and the results verified the heuristic calculations in each case.

Section 4.4 A Justification for the Heuristic Colouring Procedure

Each graph contains a critical k-chromatic subgraph. If this subgraph is coloured, and this colouring "fixed" to the subgraph, then the other vertices may have several different possible colours attached to them. Assume there exists a matrix P (with n rows and columns) such that p_{ij} is the probability that vertex i has been given the colour j for a minimal colouring of the graph.

P will not necessarily be symmetric or of any particular rank but in general will take the form



It is known that

$$\sum p_{ij} = 1 \qquad \text{(for all } i\text{)}$$

If a matrix π is formed such that

$$\pi_{jk} = \sum P^{t}_{ji} P_{ki}$$

then π_{ij} may be considered as the probability that vertex i is not joined to vertex j. It is obvious that π is a symmetric n X n matrix and that

$$0 \leq \pi_{ij} \leq 1$$

If the vertex i is a member of the critical k-chromatic subgraph of G then

$$\pi_{ii} = 1$$

otherwise

$$0 < \pi_{ii} < 1$$

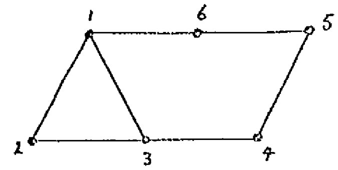
If π_{ij} is the probability that vertex i is not joined to vertex j then π bears a similarity to the matrix $\overline{A}$, and

$$\mathrm{tr}(\pi) = \sum_i \pi_{ii} \leq n = \mathrm{tr}(\overline{A}) = \sum_i \overline{A}_{ii}$$

with equality holding only when the graph on all n vertices is a critical k-chromatic graph.

It is also possible to construct a similar matrix $\overline{\pi}$ such that

$$\overline{\pi}_{ij} = 1 - \pi_{ij}$$



$A =$

	1	2	3	4	5	6	$\lambda_1 = 3.43$
1	1	1	1			1	.52
2	1	1	1				.42
3	1	1	1	1			.52
4			1	1	1		.32
5				1	1	1	.26
6	1				1	1	.32

$\overline{A} =$

	1	2	3	4	5	6	$\lambda_1 = 3.7$
1	1			1	1		.31
2		1		1	1	1	.47
3			1		1	1	.31
4	1	1		1		1	.45
5	1	1	1		1		.46
6		1	1	1		1	.45

$P =$

	1	2	3
1	1	0	0
2	0	1	0
3	0	0	1
4	1/2	1/2	0
5	3/8	1/4	3/8
6	0	1/2	1/2

$\pi =$

	1	2	3	4	5	6	$\lambda_1 = 2.0$
1	1	0	0	1/2	3/8	0	.34
2	0	1	0	1/2	1/4	1/2	.50
3	0	0	1	0	3/8	1/2	.34
4	1/2	1/2	0	1/2	5/16	1/4	.42
5	3/8	1/4	3/8	5/16	1/4	5/16	.36
6	0	1/2	1/2	1/4	5/16	1/2	.42

$\overline{\pi} =$

	1	2	3	4	5	6	$\lambda_1 = 4.7$
1	0	1	1	1/2	5/8	1	.44
2	1	0	1	1/2	3/4	1/2	.41
3	1	1	0	1	5/8	1/2	.44
4	1/2	1/2	1	1/2	11/16	3/4	.38
5	5/8	3/4	5/8	11/16	3/4	11/16	.37
6	1	1/2	1/2	3/4	11/16	1/2	.38

FIGURE 4.4.1

Showing a graph and its relation to the matrices

A, $\overline{A}$, P, π, and $\overline{\pi}$

then $\overline{\pi}_{ij}$ is the probability that vertex i is joined to vertex j. $\overline{\pi}$ will be a matrix similar to A with the provision that

$$0 \leq \overline{\pi}_{ij} \leq A_{ij} \leq 1$$

and

$$\overline{\pi}_{ij} = 1$$

if the edge (i,j) and vertices i and j are part of the critical k-chromatic subgraph of G.

The largest principal component of $\overline{\pi}$ will show a high correlation with the elements of the critical chromatic subgraph of the graph G. It is obvious that the principal eigenvector of $\overline{\pi}$ will be "similar" to the principal eigenvector of A. This explains the use of the principal eigenvector of A in the heuristic colouring procedure, and the importance of ordering the vertices by the magnitude of the elements of this vector.

The example shown in FIGURE 4.4.1 usefully illustrates the relation between G, A, A, P, π, and $\overline{\pi}$. It should be noted that the elements of the principal eigenvector of $\overline{\pi}$ bear the same relation to one another as the elements of the principal eigenvector of A.

Section 4.5 The Four Colour Problem

No work of this sort would be complete without some mention, no matter how brief, of the four colour problem. The four colour problem in graph theory and Fermat's last conjecture in number theory probably rank as the two greatest unsolved problems in mathematics.

The four colour problem is a little more than a century old. The first known source discussing the problem is a letter (dated Oct. 23, 1852) from Augustus de Morgan, Professor of Mathematics at University College London, to his friend Sir William Rowan Hamilton at Trinity College Dublin. Since that time several proofs have been proposed and each has eventually been refuted.

FIGURE 4.5.1 shows a maximally planar graph, ie. a planar graph is maximally planar if the addition of any extra edge results in the loss of the ability to represent the graph in a plane. It is easy to show (by the use of Theorem 4.3.1 and Theorem 4.1.3) that the chromatic number of such a graph is less than or equal to four.

FIGURE 4.5.2 shows a planar graph G. If the edge (1,3) is removed and the edge (2,4) inserted to form the graph G' then G' is said to have been obtained from G by a diagonal transformation. O. Ore (27) has shown that any two maximally planar graphs with

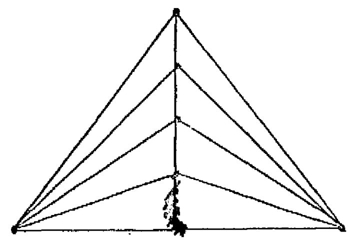


FIGURE 4.5.1

Showing a maximally planar graph

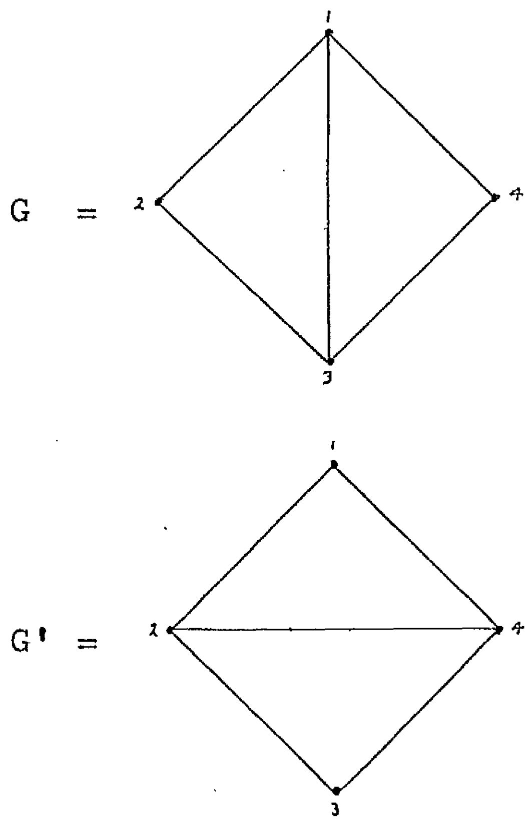


FIGURE 4.5.2

Showing a diagonal transformation

n vertices are equivalent under diagonal transformations and thus can be transformed into the form shown in FIGURE 4.5.1.

The four colour problem may now be stated as follows

Theorem 4.5.1

The chromatic number of a planar graph is invariant under diagonal transformations.

Proof

Unknown.

REFERENCES

1/ G. Ernest Anderson - <u>An Algorithm for Assigning Pupils to Classes</u>, New England School Development Council.

2/ C. Berge - <u>The Theory of Graphs and its Applications</u>, John Wiley and Sons, New York, 1962.

3/ J. F. Blakesley - <u>A Data Processing System for Scheduling University Students to Classes Using an Intermediate Speed Digital Computer</u> Thesis, Purdue University, June 1960.

4/ Busacker and Saaty - <u>Finite Graphs and Networks</u>, McGraw-Hill Inc. (1965)

5/ A. J. Cole - <u>Examination Time-tables</u>, Computer Journal (1964) p117.

6/ A. W. Colijn and M. R. Williams - <u>Scheduling Students to Classes Using Heuristic Methods</u>, Proceedings of the 5th National Conference of the Canadian Computer Society, June 1966.

7/ J. Csima and C. C. Gotlieb - <u>A Computer Method for Constructing School Time-tables</u>, University of Toronto, Canada.

8/ G. A. Dirac - <u>Map Colour Theorems</u>, Canadian Journal of Mathematics (1952) p480.

9/ G. A. Dirac - <u>Some Theorems on Abstract Graphs</u>, Proceedings of the London Mathematical Society (1952) p69.

10/ G. A. Dirac - A Short Proof of a Map Colour Theorem, Canadian Journal of Mathematics (1957) p225.

11/ P. Erdős - On the Number of Triangles Contained in Certain Graphs, Canadian Mathematical Bulliten, Vol. 7 (1964) p53.

12/ A. P. Ersov and G. I. Kozuhin - Estimates of the Chromatic Number of Connected Graphs, Doklady Akademii Nauk SSSR (1962).

13/ M. Faulkner - Some Thoughts on Computer Sectioning at Washington State University, unpublished.

14/ Ford and Fulkerston - Network Flows and Systems of Representatives, Canadian Journal of Mathematics, Vol. 10 (1958) p78.

15/ E. Forsyth and L. Katz - A Matrix Approach to the Analysis of Sociometric Data, Sociometry Vol 9 (1946) p340.

16/ H. H. Goldstone and J. Von Neumann - Principles of Large Scale Computing Machines, Collected Works of J. Von Neumann (volume 5)

17/ C. C. Gotlieb and J. Csima - The Construction of Class-Teacher Time-tables, Proceedings of the I.F.I.P. Congress (Munich) 1962.

18/ P. Hall - On the Representatives of Subsets, Journal of the London Mathematical Society (1935) p 26.

19/ F. Harrary et al - Structural Models, John Wiley and Sons (1965).

20/ F. Harrary and I. C. Ross - A Procedure for Clique Determination Using the Group Matrix, Sociometry Vol. 20 (1957) p205.

21/ R. E. Holtz - G.A.S.P. (Generalized Academic Simulation Programs), Office of the Registrar, Mass. Institute of Technology, (1964).

22/ A. G. Holzman and W. R. Turkes - Optimal Scheduling in Educational Institutions, Industrial Engineering Department, University of Pittsburgh, U.S.A.

23/ A. Kaufmann - Graphs, Dynamic Programming and Finite Games, Academic Press (1967).

24/ J. Lions - Matrix Reduction Using the Hungarian Method for the Generation of School Time-tables, Comm. A.C.M. Vol. 9 (1966) p349.

25/ J. W. Moon - On the Number of Complete Subgraphs of a Graph, Canadian Journal of Mathematics (1964)

26/ D. F. Morrison - Multivariate Statistical Methods, McGraw-Hill (1967).

27/ O. Ore - The Four Colour Problem, Academic Press (1967).

28/ J. E. L. Peck and M. R. Williams. - Examination Scheduling, Proceedings of the 1620 Users Group Conference, Western Region, June 1960, Portland U.S.A.

29/ J. Pfaltz - Unpublished private communication.

30/ R. C. Read - An Introduction to Chromatic Polynomials, Journal of Combinatorial Theory (1968) p52.

31/ G. R. Sherman - A Combinatorial Problem Arising from Scheduling University Classes - Journal of the Tennessee Academy of Sciences, Vol. 38, #3, July 1963.

32/ G. R. Sherman - Combinatorial Scheduling, Thesis University of Tennessee, U.S.A.

33/ G. Szekeres and H. S. Wilf - An Inequality for the Chromatic Number of a Graph, Journal of Combinatorial Theory, Vol. 4 (1968) p1.

R E L A T E D   M A T E R I A L

Lestooster-Samenstelling Met Elektronische Apparatuur, Rapport van de Studieconnissie Lestoosters, Stichting Studiecentrum Vor Administratieve Automatisering, Stadhouderskade 6, Amsterdam.

V. A. Abell - A Comprehensive University Scheduling System (CUSS), Proceedings of the 10th College and University Machine Records Conference, Michigan State University, May 1965.

W. W. Abendroth - How to Make Examinations Fit the Master Class Schedule, College and University Business, Vol. 32, May 1962.

M. Almond - An Algorithm for Constructing University Time-tables, The Computer Journal, Vol. 8 (1966) p331.

Appleby, Blake, and Newman - Techniques for Producing School Time-tables and their Application to Other Scheduling Problems, Computer Journal (1961) p237.

E. Barraclough - notes given at a seminar at the University of Edinburgh, Dec. 7, 1965.

E. Barraclough - The Production of School Time-tables, The Computer Journal, Vol. 8 (1965) p136.

J. Berghuis - La Composition des Horaires Scolaires, Bull Nederland - 23 Meeting of the Users Group of Gamma-Tambour Computers, Bologne.

J. B. Buxton - An Application of a Computer to the Production of Examination Time-tables, unpublished, Electronic Computing Department University of Leeds.

G. A. Dirac - Map Colour Theorems Related to the Heawood Colour Formula, Journal of the London Mathematical Society, Vol. 31 (1956) p460.

G. A. Dirac - The Structure of k-chromatic Graphs, Fundamanta Mathematicae (1953) p42.

G. A. Dirac - A Property of 4-chromatic Graphs and some Remarks on Critical Graphs, Journal of the London Mathematical Society, Vol. 27 (1952) p85.

P. Erdős and A. Hajnal - On Chromatic Number of Graphs and Set Systems, Acta Mathematica Acad. Sci. Hungaricae, Vol. 17 (1966) p61.

J. S. Folkers - A Computer System of Time-table Conditions, Thesis, Technische Hogeschool, Delft, Netherlands, (1967).

J. S. Green - Algol Programming for the KDF 9, English-Electric-Leo Computers Ltd., Kidsgrove.

F. Harrary - A Graph Theoretic Method for the Complete Reduction of a Matrix with a View Toward Finding its Eigenvalues, Journal of Mathematics and Physics, Vol. 38 (1959) p104.

F. Harrary and I. Ross - Indentification of the Liaison Persons of an Organization Using the Structure Matrix, Management Science (1955) p251.

P. J. Heawood - Map Colour Theorem, Quarterly Journal of Mathematics, (1890) p332.

N. L. Lawrie - Timetabling by Computer, Operational Analysis Note 2, Dept. of Industrial Administration, University of Strathclyde, Glasgow. 1965.

R. D. Luce and A. D. Perry - A Method of Matrix Analysis of Group Structure, Psychometrika Vol. 14 p 95.

A. R. Needham - Graph Separability and Word Grouping, Nature, Vol. 221, #5044, p105.

H. J. Ryser - Combinatorial Properties of Matrices of Zeros and Ones, Canadian Journal of Mathematics, (1957) p371.

P R O G R A M S

and

F L O W   C H A R T S

The following programs were written during the course of this investigation:

DBY005000KP4 - reads the student data from paper tape and writes it onto a magnetic tape, outputs various statistics such as the number of students in each course etc.

DBY005A00KP5 - read data from magnetic tape and produces the boolean matrix associated with the graph of the course conflicts, puts matrix onto a magnetic tape, prints out degree of each vertex

DBY005B00KP5 - Peck-Williams examination time-table procedure

DBY005C00KP5 - eigenvector approximation time-table procedure, prints out the eigenvalues and eigenvectors at each iteration

DBY005D00KP4 - build up a boolean matrix directly from paper tape data - used for investigating small graphs

DBY005E00KP5 - investigate the change of position of each vertex (in the ordering criterion) at each iteration of the eigenvector procedure

DBY005F00KP5 - determines if the large graphs may be made up of several unconnected subgraphs

DBY005G00KP5 - find the largest complete graph, prints out the boolean matrix at each iteration

DBY005H00KP7 - construct the numerical matrix associated with the graph of the student course conflicts

DBY005J00KP5 - find the largest complete graph (this is the same program as DBY005G00KP5 with large portions of the program written in USER CODE to increase its speed)

DBY005K00KP4 - read master time-table data from cards and produce time-vectors for each section, time-vectors are written to a magnetic tape

DBY005L00KP4 - produce boolean matrix corresponding to the graph of the conflicts between sections of the master time-table, matrix written to a magnetic tape

DBY005M00KP5 - produce one boolean matrix for each student indicating the conflicts in his requested course sections

DBY005N00KP5 - find all complete graphs of one particular order - large portions of the program are written in USER CODE

DBY005S00KP6 - the complete graph sectioning program, produces a great deal of output at each stage to act as diagnostic material

DBY005U00KP4 - plot graphs on a Calcomp plotter

DBY005V00KP5 - reduce graphs by ORing (ie. Theorem 4.2.1)

DBY005X00KP7 - a program which attempted to find the matrix B in relation 4.3.3 and thus determine the chromatic number of the graph

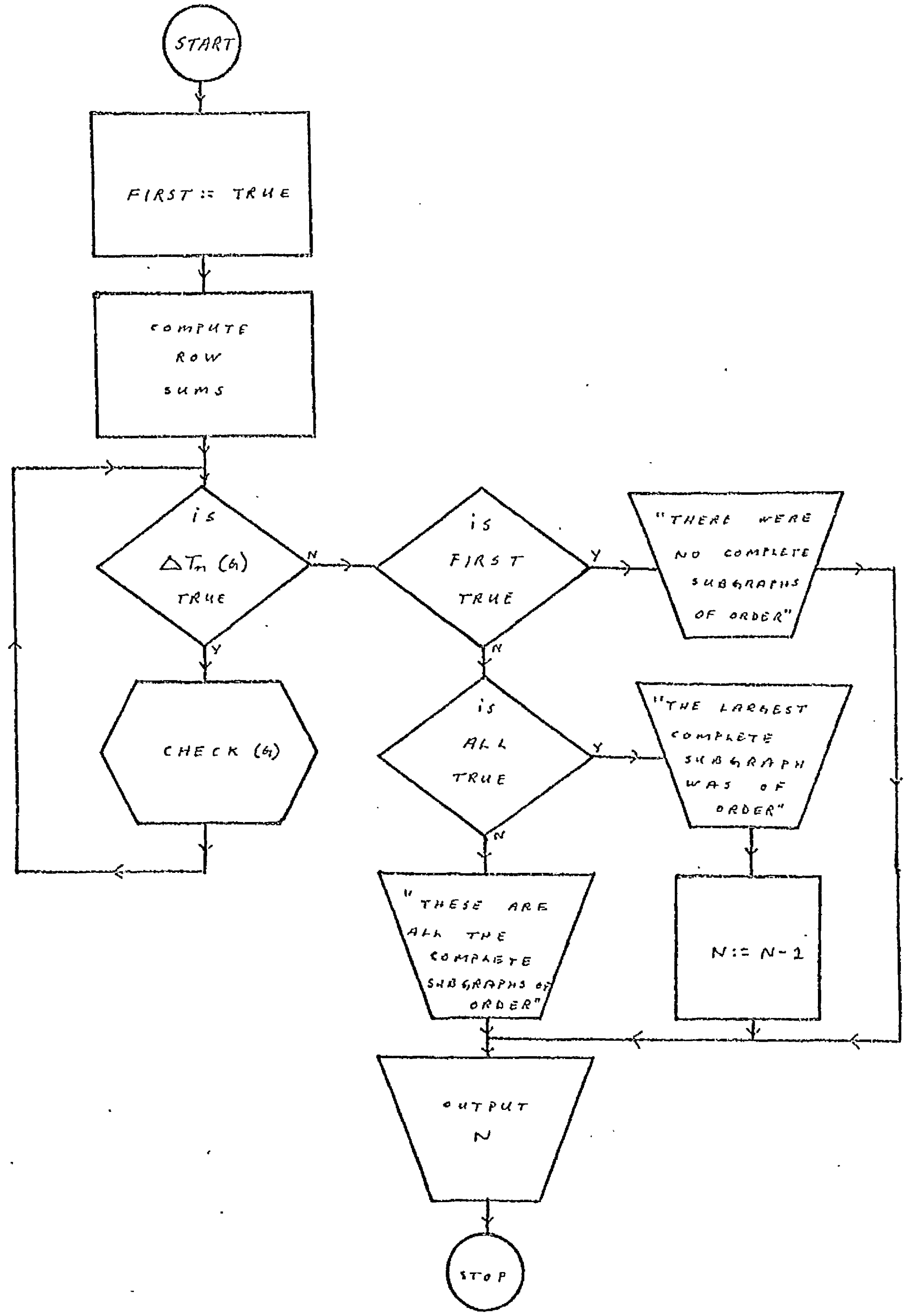
DBY005Y00KP5 - a program to produce the converse of a graph and write this converse to a magnetic tape

DBY005Z00KP4 - a program to solve Doktor Adlor's problem.

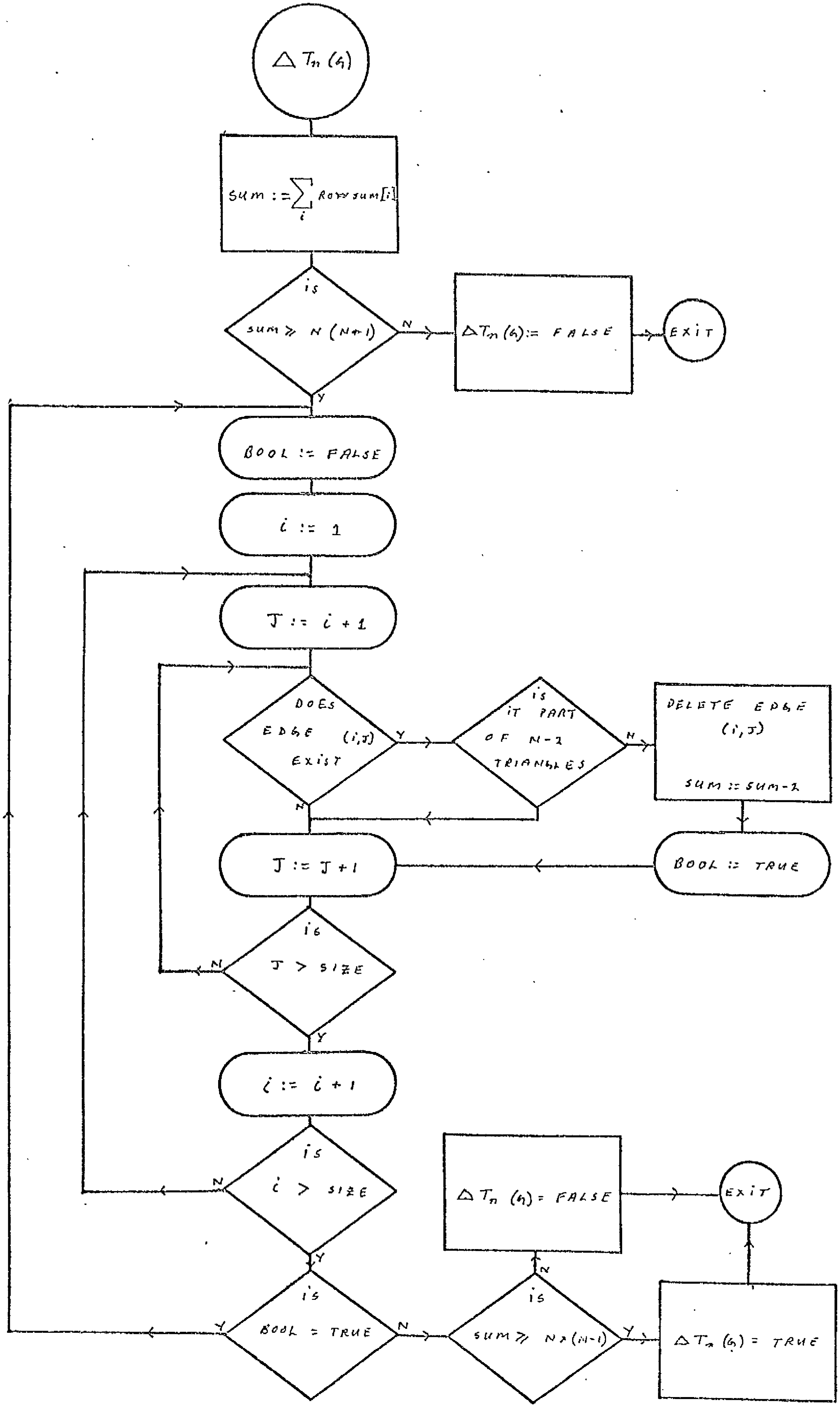
A number of other minor programs were written to do code conversions, reformatting of data, and changing the form of the data to take advantage of an increase in the computing power and peripherals of the KDF 9.

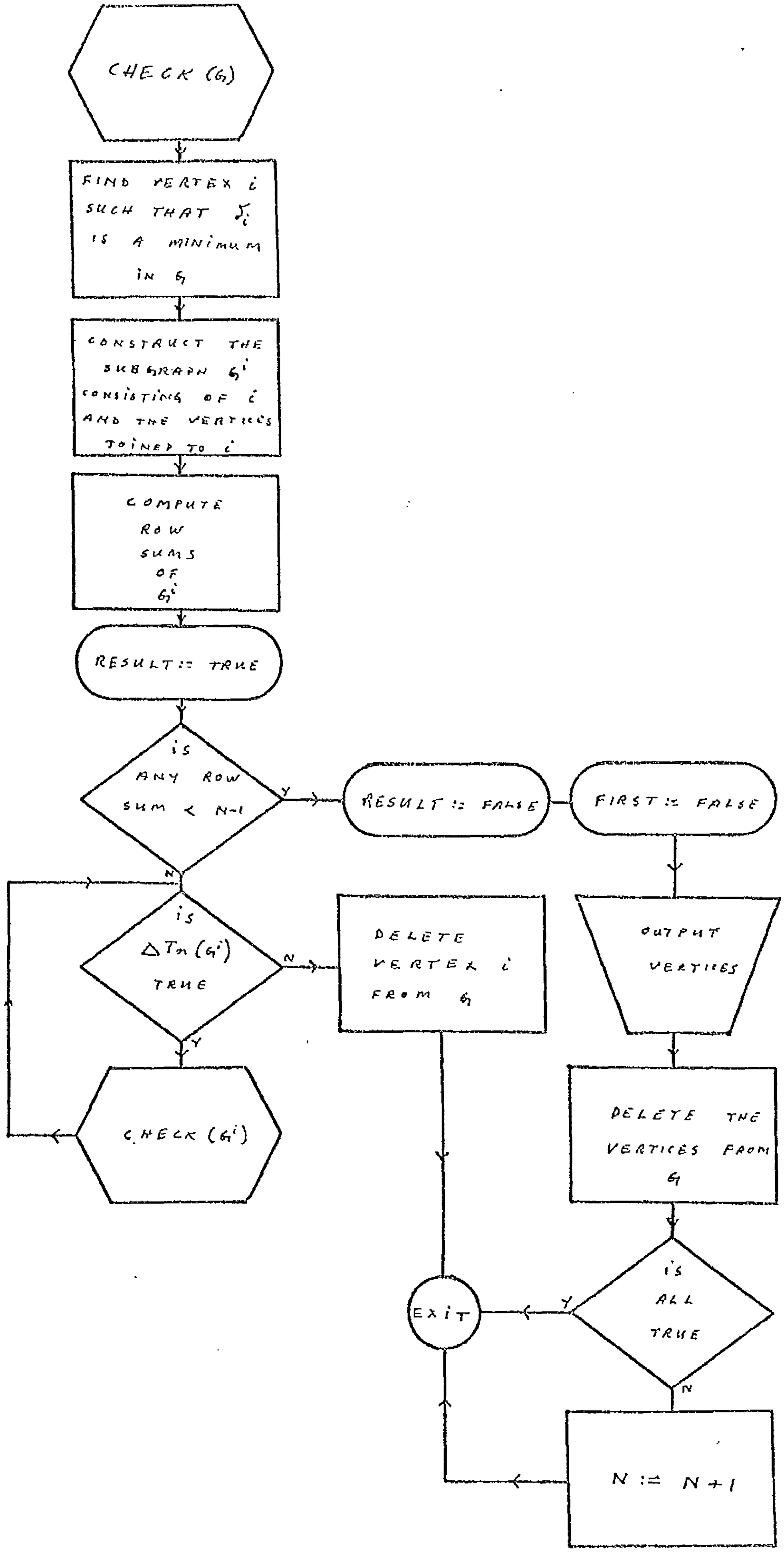
The Complete Graph Procedure

This is not a listing of the actual program used during the investigation, rather a listing of a "publication form" of the program.



START
FIRST := TRUE
COMPUTE ROW SUMS
IS $\Delta T_n (G)$ TRUE
Y
N
CHECK (G)
IS FIRST TRUE
Y
N
"THERE WERE NO COMPLETE SUBGRAPHS OF ORDER"
IS ALL TRUE
Y
N
"THE LARGEST COMPLETE SUBGRAPH WAS OF ORDER"
"THESE ARE ALL THE COMPLETE SUBGRAPHS OF ORDER"
N:= N-1
OUTPUT N
STOP


△ Tn (G)
SUM := Σ ROW SUM[i]
IS SUM ≥ N (N+1)
N
△Tn (G):= FALSE
EXIT
Y
BOOL := FALSE
i := 1
J := i + 1
DOES EDGE (i,J) EXIST
Y
IS IT PART OF N-2 TRIANGLES
N
DELETE EDGE (i,J)
SUM := SUM-2
N
J := J + 1
BOOL := TRUE
IS J > SIZE
N
Y
i := i + 1
IS i > SIZE
N
Y
△ Tn (G) = FALSE
EXIT
IS BOOL = TRUE
Y
N
IS SUM ≥ N×(N-1)
N
Y
△ Tn (G) = TRUE



CHECK (G)
FIND VERTEX i SUCH THAT δ_i IS A MINIMUM IN G
CONSTRUCT THE SUBGRAPH G^i CONSISTING OF i AND THE VERTICES JOINED TO i
COMPUTE ROW SUMS OF G^i
RESULT := TRUE
IS ANY ROW SUM < N-1
Y
N
RESULT := FALSE
FIRST := FALSE
OUTPUT VERTICES
IS $\Delta T_n(G^i)$ TRUE
N
Y
DELETE VERTEX i FROM G
CHECK (G^i)
DELETE THE VERTICES FROM G
IS ALL TRUE
Y
N
EXIT
N := N+1

```
begin
procedure outstring(dv,st);  integer dv;  string st;
    begin
    newline(70,1);  writetext(70,st);
    end  of outstring;
procedure outinteger(dv,i);  integer dv,i;
  write(70,format([sndddddd]),i);
procedure complete graphs(matrix)graph
  of:(size)vertices into:(all) complete subgraphs
  of:(order);  value size,order,all;
  integer size,order;  boolean all;  boolean array matrix;
    begin
    comment This procedure will take an undirected
      graph (in the form of a size × size symmetric
      boolean incidence matrix, whose i, jth
      element is true if and only if vertex i is
      joined by an edge to vertex j, diagonal
      elements being assumed to have the value
      false) and remove from it all edges not
      belonging to at least one complete graph of
      the required order, where order ≥ 3. The
      procedure may be used to determine all the
      complete subgraphs of a particular order by
      setting the variable all to the value true,
```

or may be used to find the order of the largest complete subgraph by setting the variable all to the value *false*. If the procedure is attempting to find the order of the largest complete subgraph it will first attempt to find a complete subgraph of the initial order and, if successful, will then increase the parameter order by one and try again, the vertices corresponding to one complete subgraph of the appropriate order are output at each iteration. The procedure is based upon the fact that each edge in a complete graph of order m is a member of m-2 edge circuits of length 3. The procedure contains its own output statements but the input is considered to come from the parameter list.;

```
integer i;
integer array row,row number[1:size];
boolean first,combfirst;
procedure delete(matrix,size,i,row);
  value size,i;  boolean array matrix;
  integer array row;  integer size,i;
  comment this procedure will set the elements
  in the i th row and column of the boolean
  array matrix to the value false and update
```

```
  the vector row to indicate the new number
  of true elements in each row of the boolean array;
  begin
  integer j;
  for j := 1 step 1 until size do
    if row[j]>0 then
      begin
      if matrix[i,j] then
        begin
        matrix[i,j]:=matrix[j,i]:=false;
        row[j]:=row[j]-1;
        end ;
      end of deleting row and column i;
  row[i]:=0;
  end of the procedure delete;
procedure zero(matrix,size); value size;
  integer size; boolean array matrix;
  comment this procedure will assign the value
  false to each element of the boolean array matrix;
  begin
  integer i,j;
  for i := 1 step 1 until size do
    for j := 1 step 1 until size do
      matrix[i,j]:=matrix[j,i]:=false;
  end of procedure zero;
```

```
procedure comb(n,r,i); integer n,r; integer array i;
  comment This procedure is a modified version
  of Algorithm 154. The distinct combinations
  of the first n integers taken r at a time
  are generated in i in lexicographical order
  starting with an initial combination of the
  r integers 1,2,.....r. The boolean variable
  combfirst is nonlocal to comb and must be
  true before the first call, thereafter it
  remains false until all combinations have
  been generated;
  begin
  integer s,j;
  if combfirst then
      begin
      for j := 1 step 1 until r do i[j]:=j;
      combfirst:=false; goto exitcomb;
      end of initial combination;
  if i[r]<n then
      begin
      i[r]:=i[r]+1; goto exitcomb;
      end ;
  for j := r-1 step -1 until 1 do
    if i[j]<n-r+j then
      begin
```

```
        i[j]:=i[j]+1;
        for s := j+1 step 1 until r do i[s]:=i[j]+s-j;
        goto exitcomb;
        end  ;
     combfirst:=true;
exitcomb:
     end  of the procedure comb;
    integer procedure two row sum(i,j,matrix,size);
     value i,j,size;  integer i,j,size;
     boolean array matrix;
     comment this procedure will return the number
     of true elements in a vector formed by a
     boolean AND operation between the ith and
     jth rows of the boolean array matrix;
     begin
     integer sum,k;
     sum:=0;
     for k := 1 step 1 until size do
       if matrix[k,i] and matrix[k,j] then sum:=sum+1;
     two row sum:=sum;
     end  of procedure two row sum;
    integer procedure row sum(i,matrix,size);
     value i,size;  integer i,size;  boolean array matrix;
     comment this procedure will return the number
     of true elements in the ith row of the
```

```
boolean array matrix;
begin
integer sum,k;
sum:=0;
for k := 1 step 1 until size do
  if matrix[k,i] then sum:=sum+1;
row sum:=sum;
end  of procedure row sum;
boolean procedure remove edges(matrix,size,row,order);
value size,order;  integer size,order;
integer array row;  boolean array matrix;
comment this procedure will remove from the
graph (defined by the array matrix) all
edges which are not members of at least
order-2 edge circuits of length three. When
the removal is complete the elements of the
matrix are checked, if there are enough
edges left to make a complete graph of the
required order then the procedure has the
value true else false;
begin
integer sum,i,j;
boolean finished;
sum:=0;
for i := 1 step 1 until size do sum:=sum+row[i];
```

```
      comment check to see if there are enough
        edges at the outset;
      if sum≥order×(order-1) then goto work;
      remove edges:=false;  goto quit;
work:  finished:=true;
      for i := 1 step 1 until size do
        if row[i]>0 then
          for j := i+1 step 1 until size do
          if row[j]>0 then
          begin
          if matrix [i,j] then
            begin
            if two row sum(i,j,matrix,size)<order-2 then
                begin
                comment delete the edge between
                  vertex i and vertex j;
                matrix[i,j]:=matrix[j,i]:=false;
                row[i]:=row[i]-1;  row[j]:=row[j]-1;
                sum:=sum-2;  finished:=false;
                end  of deleting an edge;
            end  of checking one row;
          end  of the entire matrix;
      if not finished then goto work;
      remove edges:=not sum<order×(order-1);
quit:
```

```
  end  of procedure remove edges;
procedure check(matrix,row,row number,size,order);
  integer size,order;
  integer array row,row number;  boolean array matrix;
  begin
  boolean result;
  integer i,j,k,l,m;
  comment this procedure will check that the
    vertices and edges of the graph are, in
    fact, part of a complete graph of the
    required order. The procedure starts by
    finding the vertex with the smallest
    nonzero degree, then determines the
    complete graphs attached to this vertex.;
  k:=size;  i:=0;
  for j := 1 step 1 until size do
    if row[j]>0 and row[j]<k then
      begin
      i:=j;  k:=row[j];
      end  of finding a vertex for further
      investigation;
  k:=k+1;
  comment now investigate the (possible)
    complete subgraph on vertex i;
      begin
```

```
boolean array subgraph[1:k,1:k];
integer array new row,new row number[1:k];
zero(subgraph,k);  new row number[1]:=i;  l:=2;
for m := 1 step 1 until size do
  if row [m]>0 then
  begin
  if matrix[i,m] then
      begin
      new row number[l]:=m;  l:=l+1;
      end  ;
  end  ;
for l := 1 step 1 until k do
  for m := l step 1 until k do
  subgraph[l,m]:=subgraph[m,l]:=matrix[new row
  number[l],new row number[m]];
comment a subgraph has just been
  constructed. It consists of vertex i
  and those vertices joined to i by an
  edge, along with any edges joining
  the selected vertices. The procedure
  now checks to see if this is a
  complete graph.;
result:=true;
for m := 1 step 1 until k do
  begin
```

```
new row[m]:=row sum(m,subgraph,k);
if new row[m]<k-1 then result:=false;
new row number[m]:=row number[new
  row number[m]];
comment this keeps our vertex
  numbering system constant;
end  of elementary check;
if result then goto end of procedure;
comment the elementary check was not
  sufficient to determine if this is a
  complete graph.;
again:  if remove edges(subgraph,k,new row,order) then
  begin
  check(subgraph,new row,new row
    number,k,order);
  goto again;
  end  of detailed check;
comment the subgraph under check was
  not complete. Row and column i of the
  graph in the next highest level of
  recursion must be set to false.;
delete(matrix,size,i,row);  goto exit;
end of procedure:  first:=false;
  begin
integer array v[1:order],sgvertex[1:k];
```
integer array v[1:order],sgvertex[1:k];

```
             combfirst:=true;
combinations:  comb(k,order,v);
             if combfirst then goto last;
             outstring(1,[found*a*complete*
               graph*of*order]);
             outinteger(1,order);
             outstring(1,[which*consists*of*the*
               following*vertices]);
             for m := 1 step 1 until order do
               outinteger(1,new row number[v[m]]);
             if all then goto combinations else
               goto exit1;
      last:  sgvertex[1]:=i;  m:=2;
             for j := 1 step 1 until size do
               if matrix[i,j] then
                 begin
                 sgvertex[m]:=j;  m:=m+1;
                 end  ;
             j:=0;
             for l := 1 step 1 until k do
               if row[sgvertex[l]]=k-1-j then
                 begin
                 delete(matrix,size,sgvertex[l],row);
                 j:=j+1;
                 end  ;
```

```
               end  of output;
             end  of subgraph checking;
exit1:   if not all then
             begin
             first:=true;
             if k>order+1 then order:=k else order:=order+1;
             comment this ensures that if the
               procedure has found a complete graph
               of order k it does not keep looking
               for complete graphs of order less than k;
             goto once more;
             end  ;
exit:
       end  of procedure check;
     comment this is the start of the main procedure;
     first:=true;
     for i := 1 step 1 until size do
       begin
       comment row number[i] is the name of the
         i th vertex;
       row[i]:=row sum(i,matrix,size);  row number[i]:=i;
       end  of setting up data vectors;
once more:  if remove edges(matrix,size,row,order) then
       begin
       check(matrix,row,row number,size,order);
```

```
          goto once more;
          end  of graph check;
        if not first then
          begin
          if all then
            outstring(1,[these*are*all*the*complete*
            subgraphs*of*order]) else
              begin
              outstring(1,[the*largest*complete*
                subgraph*was*of*order]);
              order:=order-1;
              end  ;
          end
          else
          outstring(1,[there*were*no*complete*subgraphs*of*
          order]);
        outinteger(1,order);
        end  of procedure complete graphs;
      integer size,order,i,j;
      boolean all;
start:  open(70);  open(20);  newline(70,5);
      outstring(1,[M.R.WILLIAMS-COMPUTING]);  size:=read(20);
      order:=read(20);  all:=read boolean(20);
      outstring(1,[size**order**all]);  outinteger(1,size);
      outinteger(1,order);
```

```
    if all then outstring(1,[true]) else outstring(1,[false]);
       begin
       boolean array graph[1:size,1:size];
       for i := 1 step 1 until size do
         for j := 1 step 1 until size do graph[i,j]:=false;
input:   i:=read(20);
       if i=999 then goto end of read;
       j:=read(20);  graph[i,j]:=graph[j,i]:=true;  goto input
       end of read:
         close(20);
       complete graphs(graph,size,all,order);  close(70);
       goto start;
       end  ;
    end→
```

The Peck-Williams Examination Time-table Procedure

Communications of the ACM

Volume 9 / Number 6 / June, 1966

ALGORITHM 286
EXAMINATION SCHEDULING [ZH]

J. E. L. Peck and M. R. Williams (Recd. 17 Mar. 1964, 25 Jan. 1965 and 1 Mar. 1966)
University of Alberta, Calgary, Alta., Canada

procedure *partition* (*incidence*) graph of order : (*m*) into : (*n*) parts using weights : (*w*) bound : (*max*) preassignment : (*preassign*) of number : (*pren*);
Boolean array *incidence*; **integer array** *w*, *preassign*;
integer *m*, *n*, *max*, *pren*;

comment This is an heuristic examination time-tabling procedure for scheduling *m* courses in *n* time periods. It is essentially the problem of graph partitioning and map coloring.

In the terminology of graph theory: Given a graph of *m* vertexes with a positive integer weight $w[i]$ at the ith vertex, partition this graph into no more than *n* disjoint sets such that each set contains no two vertexes joined by an edge, and such that the total weight of each set is less than the prescribed bound *max*.

We represent the graph as an $m \times m$ symmetric Boolean matrix *incidence* whose i,jth element is **true** if and only if vertex i is joined to vertex j by an edge (if a student is taking both course i and course j), diagonal elements being assigned the value **true**. The weight assigned to the ith vertex (number of students in the ith course) is $w[i]$. We shall see below that preassignment is permitted. The number of courses to be preassigned is given in *pren* and the course *preassign* $[i, 1]$ is to be placed at the time *preassign* $[i, 2]$.

This procedure does not minimize the second order incidence i.e. a vertex i being assigned to the set k, where the set $k-1$ contains a vertex j joined to i (a student writing two consecutive examinations), but this may be done by rearranging the sets after the partitioning is completed. The procedure contains its own output statements, but its driver should provide the input;

begin integer array *row* [1:*m*], *number* [1:*n*];
  integer *i*, *j*, *sum*, *course*, *time*;
  Boolean *preset*, *completed;*
INITIALIZE: *preset*:= **false**;
  for *j* := 1 **step** 1 **until** *n* **do** *number* [*j*] := 0;
  for *i* := 1 **step** 1 **until** *m* **do**
  begin *sum* := 0;
    for *j* := 1 **step** 1 **until** *m* **do**
    if *incidence* [*i*, *j*] **then** *sum* := *sum* + 1;
    row [*i*] := *sum*
  end *INITIALIZE*. Note that *row* [*i*] now contains the multiplicity of, or number of edges at the vertex i (number of courses which conflict with the course i). Of course since the incidence matrix is symmetric, less than half ($i > j$) need be stored. However, this procedure, for the sake of simplicity, is written for the whole matrix. Also note that *row* [*i*] will eventually contain the negative of the set number to which the ith vertex is assigned (examination time for the ith course) and *number* [*j*] will contain the weight of the jth set (number of candidates at time j). From here on we drop the allusions to graph theory in the comments;
THE PREASSIGNMENT: **for** *j* := 1 **step** 1 **until** *pren* **do**
  begin comment preassignment of courses to times is now car-

```
        ried out. If pren = 0, then there are no preassignments;
      course := preassign [j,1]; time:= preassign [j,2];
      comment We now attempt to assign this course to the given
        time;
SCRUTINIZE: if row [course] < 0 then
      begin outstring (1, 'This course'); outinteger (1, course);
        outstring (1, 'is already scheduled at time');
        outinteger (1, -row[course]);  go to NEXT
      end;
      if number [time] + w[course] > max then
      begin outstring (1, 'Space is not available for course');
        outinteger (1, course); outstring (1, 'at time');
        outinteger (1, time);  go to NEXT
      end;
      for i := 1 step 1 until m do
        if row [i] = - time then
        begin if incidence [i, course] then
          begin outstring (1, 'course number');
            outinteger (1, course); outstring (1, 'conflicts with');
            outinteger (1,i);
            outsiring (1, 'which is already scheduled at');
            outinteger (1, time),
            go to NEXT
          end if incidence
        end if row;
SATISFACTORY: row[course] := -time;
      number [time] := number [time] + w [course];
      preset := true;
NEXT:
    end THE PREASSIGNMENT;
MAIN PROGRAM: begin Boolean array available [1:m];
      integer next;
      procedure check (course); integer course;
      begin integer j; comment This procedure renders un-
            available those courses conflicting with the given course;
        for j := 1 step 1 until m do
          if incidence [course,j] then available [j] := false
      end of procedure check.
        For each of the n time periods we select a suitable set of non-
        conflicting courses whose students will fit the examination
        room;
START OF MAIN PROGRAM:
      for time := 1 step 1 until n do
        if preset ≡ number[time] > 0 then
        begin comment The preceding Boolean equivalence di-
              rects the attention of the program initially only to
              those times where prescheduling has occurred. We now
              determine the available courses (i.e. unscheduled and
              nonconflicting). If course i is already scheduled, then
              row[i] is negative;
          completed := true;
          for i := 1 step 1 until m do if row [i] > 0 then
          begin available [i] := true; completed := false end
            else available [i] := false;
          if completed then go to OUTPUT;
          if preset then
          begin comment Some courses were prescheduled at
                this time. It is necessary to render their conflicts un-
                available;
            for i := 1 step 1 until m do
              if row[i] = -time then check (i)
```

```
            end prescheduled courses.
              We now select the available course with the most con-
              flicts. This is essentially the heuristic step and there-
              fore the place where variations on the method may be
              made;
AGAIN:
            sum := 0;
            for i := 1 step 1 until m do
              if available [i] ∧ row [i] > sum then
              begin next := i;  sum := row [i] end most conflicts;
            if sum > 0 then
            begin comment  There exists an available course, so
                we test it (viz next) for size. If it does not fit we look
                for another;
              available [next] := false;
              if number [time] + w[next] > max then go to AGAIN;
              comment  If we are here the course will fit so we use it;
              row [next] := −time;
              number [time] := number [time] + w[next];
              check (next); go to AGAIN
            end sum > 0
        end of the time loop;
        if preset then
          begin preset := false; go to START OF MAIN
            PROGRAM end
            In case of prescheduling this takes us back to try the re-
            maining time periods.
              If we have reached here with completed true then all
            courses are scheduled, but the converse may not be true,
            therefore;
        if ¬ completed then
        begin completed := true;
          for i := 1 step 1 until m do
            if row [i] > 0 then completed := false
        end ¬ completed and
      end of the main program;
OUTPUT: if ¬ completed then
      begin comment The following for statement outputs the
          courses that were not scheduled;
        outstring (1, 'courses not scheduled');
        for i := 1 step 1 until m do
          if row [i] > 0 then outinteger (1,i)
      end not scheduled.
        The following outputs the time period j, the number of stu-
        dents number[j] and the courses i written at time j;
TIMETABLE: outstring(1, 'time  enrolment  courses');
      for j := 1 step 1 until n do
      begin outinteger (1,j); outinteger (1, number[j]);
        for i := 1 step 1 until m do
          if row[i] = −j then outinteger (1,i)
      end j.
        The following outputs the courses, the times at which they are
        written, and their enrolment;
      outstring (1, 'course  time  enrolment');
      for i := 1 step 1 until m do
        if row [i] < 0 then begin outinteger (1, i); outinteger (1, row [i]);
          outinteger (1, w[i]) end
        else
        begin outinteger(1,i); outstring(1, 'unscheduled');
          outinteger (1, w[i])
        end
    end of the procedure
```

Eigenvector Approximation Procedure

All the boolean matrices were kept in the core store of the KDF 9 by storing one boolean element per bit (ie. 48 boolean elements could be stored in each KDF 9 word). This form of storage required the use of USER CODE procedures for bit interrogation and manipulation. The matrices were stored by rows in such a manner that each row occupied an integral number of words, any excess bits being set to the value <u>false</u>. The instruction set of the KDF 9 makes certain bit manipulations easy to code and efficient to perform, for example the procedure "eigen" will perform a matrix-vector multiply (700 X 700) in just under 7 seconds. "eigen" is included here as being typical of the USER CODE procedures used.

```
procedure eigen(row,row2,matrix,size);
      value size;  integer size;
      real array matrix, row, row2;
      comment this procedure will take an approximation
            to the largest eigenvector of MATRIX (given
            in ROW) and leave a closer approximation
            in ROW2. MATRIX is a square 0-1 bit matrix
            of size SIZE.;
      KDF9 4/6/0/0;
      [size]; DUP; SET48; ÷I; ERASE; SET1; +; DUP;
      (number of words per row);
      =RC10; =RI13; SET1; +; =RC14; (number of rows);
      SET48; =RC11; (number of bits per word);
      [row]; =M11; M11; SETAY0; +; =M11;
      (address of the start of ROW);
      [row2]; =M14; M14; SETAY0; +; =M14;
      (address of the start of ROW2);
      [matrix]; =M13; M13; SETAY0; +; =M13;
      (address of MATRIX[0,0]);
      3; ZERO; Q11TOQ15; Q10TOQ12; (set counters to
      deal with one row of matrix);
      2; M13M12Q; SET48; =C15; (set up counters to
      deal with one word of the current matrix row);
```

```
*1; ZERO; SHLD1; NEG; MOM15Q; AND; CAB; +F; REV; J1C15NZS;
(that is the inner loop (short loop jump));
ERASE; J2C12NZ; (get next word of current matrix row);
=MOM14Q; M+I13; (get next row of matrix);
J3C14NZ; (is job finished);
EXIT;
ALGOL;
```