AN ALGEBRAIC APPROACH TO THE GRAPH ISOMORPHISM PROBLEM

by

DIANE MARIE BOWMAN

A THESIS SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

AT THE

COMPUTING LABORATORY

UNIVERSITY OF NEWCASTLE UPON TYNE

ENGLAND

JUNE 1977

ABSTRACT

This thesis is addressed to the problem of determining graph isomorphism on the computer by exploiting the algebraic properties of the adjacency matrices of the given graphs. More specifically, some results are proven which take advantage of the information provided by the eigenvectors of the adjacency matrices in order to determine graph isomorphism. The proposed methods are capable of either detecting isomorphism or proving non-isomorphism for any given pair of n-point cospectral graphs that are connected and without loops or multiple edges. The most significant contribution of the work is that it provides a means for dealing with cospectral graphs that have highly multiple eigenvalues.

In Chapter 1 a review of previous isomorphism algorithms is given along with some preliminary notation and definitions. In Chapter 2 an algorithm is proposed which will detect non-isomorphism in order $n^3 \log n$ time at worst for many non-isomorphic cospectral graphs (including some with highly multiple eigenvalues). In Chapter 3 the algorithm incorporates backtracking to handle isomorphic graphs and the "more difficult" cases of non-isomorphic graphs including strongly regular graphs and graphs of designs. As yet we have not discovered a pair of non-isomorphic cospectral graphs that require more than n "top level" backtracks from this algorithm. For the graphs that require n backtracks, the predicted computation time to prove non-isomorphism is of order n^6. Alternatively, the algorithm determines isomorphism of the "non-difficult" class of graphs using 0 backtracks in n^5 time at worst. In Chapter 4 a modification of the backtracking algorithm is given for determining the generators of the automorphism group of a graph. Chapter 5 contains a study of computer timings of the algorithm which confirms that the predicted bounds are in fact too high as n grows. Finally, some directions for future research are suggested in Chapter 6.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor, Dr. C. R. Snow for his sound advice, suggestions, and constant encouragement throughout the course of the research for this thesis.

Special thanks are due to Mr. J. S. Clowes for many spirited discussions of this problem. His mathematical insight, endless flow of suggestions, and enthusiasm contributed greatly to the completion of this thesis.

I am grateful to both Dr. Snow and Mr. Clowes for their detailed and constructive criticism of the various versions of the manuscript and for their persistent speed in reading of the chapters in spite of their own busy schedules.

I wish to thank Dr. A. K. Dewdney and Dr. A. Mowshowitz who were responsible for awakening my interest in the graph isomorphism problem.

Thanks are also due to the girls in the Computing Laboratory office for their kind assistance and to the night operators for their friendly and cheerful disposition in the early hours of the morning.

Finally I am very grateful to the members of the Association of Commonwealth Universities and of the British Council for their assistance in many matters and for making it possible for me to study at Newcastle University.

The research in this thesis was funded by a Commonwealth Scholarship which was gratefully received from the Commonwealth Scholarship Commission in the United Kingdom.

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.0 Introduction

This thesis is addressed to the problem of determining whether or not two graphs are isomorphic in a polynomial bounded time. This rather famous graph theoretical problem has general application in the fields of Structural Chemistry, Network Theory, Information Retrieval, Artificial Intelligence, Linguistics, and Computer Science for model representation through the use of graphs. In particular, this problem is of interest to Computer Scientists because of the challenge and practical need to develop algorithms which can be utilized to generate graphs (even on relatively few points) in reasonable time on the computer. As D. Corneil [1974] points out:

> "To a mathematician this problem is somewhat uninteresting since if the given graphs are finite, then all the possible mappings can be computed and checked for isomorphism in finite time, namely proportional to n! where n is the order of each graph. When the obvious method was used on a computer one soon found that the amount of time needed to determine isomorphism of fairly small graphs was astronomical. It was typical to see the following type of argument: 'It is easy to see that even with an extremely fast present day computer, at least $10^6 x \mu^2$ seconds are required to perform one reordering of the nodes of G_1 and to compare the graph determined by this reordering with G_2. Thus a method based on examining all possible node reorderings may require as much as 6 minutes for two 10-node graphs, 9 years for two 15-node graphs, and $3x10^5$ centuries for two 20-node graphs. For graphs with more than ten nodes, a more practical procedure is necessary.'"

Graph theorists require sets of non-isomorphic graphs to study properties of graphs and to search for counter-examples to conjectures.

Toward this end, it is desirable to be able to directly generate on the computer a set of all the non-isomorphic graphs on n points with q lines. However, due to the enormous computing time required for such a task with present methods, the largest known complete set of graphs on n points is for n = 9 (see Baker et al. [1974]). In the computer generation of graphs, it is the isomorphism determination part of the method which is the source of the enormous computation time. Hence, computer generation of graphs on n points for $n > 9$ may be practically feasible if the computation time of the isomorphism determination grows algebraically rather than exponentially with n.

In this Chapter we briefly review the major previous isomorphism algorithms. In doing so it becomes apparent that there are many prerequisite conditions for graph isomorphism. Ideally one would like to discover an isomorphism invariant property which could be computed in polynomial time and which is sufficient to distinguish between non-isomorphic graphs. However, falling short of this ideal, it is important to consider which combination of these many isomorphism invariant conditions frequently determines the difference between two non-isomorphic graphs and at the same time requires a polynomial computation of low order.

The spectrum of a graph is one such isomorphism invariant property: the <u>spectrum</u> of a graph G is defined to be the set of numbers which are the eigenvalues of the adjacency matrix A of G together with their multiplicities as eigenvalues of A. If the distinct eigenvalues are: $\alpha_1 < \alpha_2 < \ldots\ldots < \alpha_k$ then their corresponding multiplicities are $m(\alpha_1)$, $m(\alpha_2)$, ..., $m(\alpha_k)$. Two graphs are said to be <u>cospectral</u> if they have the same eigenvalues with the same multiplicities. Although the spectrum of a graph is an isomorphism invariant property it is not a sufficient

condition for isomorphism. In spite of Turner's [1967 & 1968] pessimism at solving the graph isomorphism problem by spectra, the research in this thesis focuses on the spectral related properties of a graph. In particular, we will investigate how the eigenvectors associated with the eigenvalues of the adjacency matrix can be used to discern the difference between a pair (or more) of non-isomorphic cospectral graphs. Such an approach enables us to process many "worst case" types of graphs for previous isomorphism algorithms in $n^3 \log n$ time. In evaluation of the proposed method, we shall examine how it successfully handles various cospectral examples of trees, regular graphs, strongly-regular graphs, and graphs of balanced incomplete block designs which have been a source of great difficulty if not failure to other algorithms.

In 1961 at a meeting of the American Mathematical Society, F. Harary [1962] mistakenly conjectured that the cospectrality of two graphs was a sufficient condition for isomorphism. However, it is well known that the conjecture has been disproved by a number of counter-examples, some of which are due to Collatz and Sinogowitz [1957], R.C. Bose [1961], R.H. Bruck [1963], A.J. Hoffman [1963], G. Baker [1966], J. Turner [1967 & 1968], and M. Fischer [1966]. Even though the spectrum does not uniquely determine a graph, it has the advantage of frequently distinguishing between two arbitrary graphs (on the same number of points) that would otherwise seem very similar. Harary et al. [1971] give evidence to support this statement in their findings that the probability that a pair of graphs chosen at random from the collection of non-isomorphic connected ones with p nodes are cospectral, is given by f(p) where $f(6) = 0.82 \times 10^{-4}$ and $f(7) = 0.61 \times 10^{-4}$. They remark that it is tempting to conjecture that f(p) is decreasing for $p \geq 6$. Similarily for trees, they found that if g(p) denotes the probability that a pair of non-isomorphic p-node trees

are cospectral, then:

$$g(p) = \begin{cases} 0 & \text{if } p \leq 7 \\ 0.40\text{x}10^{-2} & \text{if } p = 8 \\ 0.46\text{x}10^{-2} & \text{if } p = 9 \\ 0.72\text{x}10^{-3} & \text{if } p = 10 \end{cases}$$

A further point in favour of a spectral approach to this problem is that the computation time for eigenvalues and eigenvectors of an n-point graph is of order n^3 (see Gourlay & Watson [1973] and Reinsch and Wilkinson [1971]). In view of present algorithms, this most certainly falls within the computation time that one would consider efficient for the graph isomorphism problem.

Having decided to use the spectral properties of the graph to determine isomorphism, we are left to contend with the pairs of cospectral graphs. The central theme of this thesis is therefore to investigate and demonstrate the efficient and effective use of spectral information in determining whether or not two cospectral graphs are isomorphic. Some preliminary notation and definitions are presented in the next section. In the rest of this Chapter we shall define what is meant by an 'efficient' algorithm and review previous isomorphism algorithms.

1.1 Notation and Definitions

The notation adopted in this thesis is largely that of Harary [1969]. Definitions needed for the whole discussion are given in this section; others will be introduced as necessary. Unless otherwise stated, the term graph will mean one that is finite, undirected, with no loops or parallel edges. In addition the graphs will most often be considered as labeled (see defn. below) and connected.

A graph $G = (V,E)$ is a finite non-empty set $V = V(G)$ with a collection $E = E(G)$ of unordered pairs of distinct elements of $V(G)$. The

elements of V are called the points, vertices, or nodes of G. Each pair $uv = \{u,v\}$ in E(G) is called a line (or edge) of G and the given points u,v in V(G) are said to be adjacent. A point u in V(G) is incident with a line x in E(G) if $x = uv$ for some v in V(G).

The degree of a point v of G (denoted deg(v)) is the number of lines in E(G) incident with v. A graph is regular of degree d if all points in V(G) are of degree d.

Let $V(G) = \{v_1, v_2, \ldots, v_n\}$ be the set of points of a graph G. Let $D = (d_1, d_2, \ldots, d_n)$ where each $d_i = \deg(v_i)$. Rename the elements of D so that $d_1 \geq d_2 \geq \ldots \ldots \geq d_n$. Then the n-tuple D is called the degree sequence of the graph G.

A bipartite graph G is a graph such that V(G) can be partitioned into two subsets V_1 and V_2 such that every line in E(G) is of the form $x = ab$ where $a \in V_1$ and $b \in V_2$.

The complement $\bar{G}$ of a graph G on n points has $V(\bar{G}) = V(G)$ as its set of points; the set of edges $E(\bar{G})$ is determined as follows: let $E^*(G)$ be the set of all possible unordered pairs of distinct points of V(G), then $E(\bar{G}) = E^*(G) - E(G)$.

A graph H is called a subgraph of a graph G if $V(H) \subset V(G)$ and $E(H) \subset E(G)$ such that for every $x = uv$ in E(H), this implies that $u,v \in V(H)$ and $x \in E(G)$. Such a subgraph H is called a spanning subgraph if $V(H) = V(G)$.

A labelling φ of a graph G with n points is a bijection of an n-element set to V(G). A graph G together with a labelling φ is said to be labeled. Most often we will use an ordered set of labels to represent V(G). Let G be a labeled graph with vertices $v_1, v_2, \ldots, v_n$. Then the adjacency matrix $A(G) = [a_{ij}]$ of G is an nxn binary matrix in which $a_{ij} = 1$ if v_i is adjacent to v_j and $a_{ij} = 0$ otherwise.

An isomorphism of a graph G to a graph H is a bijection f of V(G) onto V(H) which preserves adjacencies. Two graphs G and H are said to be

isomorphic if there exists an isomorphism between them. An automorphism of a labeled graph G is an isomorphism of G to itself. It is easy to see that an automorphism may be regarded as a permutation of the set of labels which preserves adjacencies. Moreover, it is trivial to verify that the collection of automorphisms of G (denoted Γ_G) forms a group called the automorphism group of G. If σ is a permutation on the set $\{1,2,3,\ldots,n\}$ then the $\{1,2,3,\ldots,n\}$ may be split up into disjoint classes called the orbits of σ by the following equivalence relation: each class consists of elements which can be carried into each other by some power of σ (the classes consist of the elements in the disjoint cycles of σ). The concept of an orbit of a permutation σ may be extended to an orbit of a group of permutations Γ defined on a set S. An equivalence relation can be defined on S so that two elements are Γ-equivalent provided that one can be mapped onto the other by some member of Γ. The resulting equivalence classes of S are called the orbits of the permutation group Γ. The automorphism partitioning of a graph G is the collection of orbits of its automorphism group.

A graph G is (vertex) transitive if for any two vertices $x,y \in V(G)$, there exists at least one automorphism of G mapping x onto y. A subgraph H of G is a transitive subgraph of G if for any two nodes, $x,y \in E(H)$ there exists an automorphism of G mapping x onto y. A graph G is rigid if the only member of Γ_G is the identity.

A walk of a graph G is an alternating sequence of points and lines $v_0,x_1,v_1,\ldots\ldots,v_{n-1},x_n,v_n$ beginning and ending with points of V(G), in which each line is incident with the two points immediately preceding and following it. This walk joins v_0 and v_n and is sometimes called a v_0-v_n walk. It is closed if $v_0 = v_n$ and is open otherwise. It is a trail if all the lines are distinct and a path if all the points (and necessarily

all the lines) are distinct. If the walk is closed, then it is a cycle provided its n points are distinct and $n \geq 3$ (sometimes denoted an n-cycle). An n-cycle of a graph G is called a Hamiltonian cycle if $V(G) = n$. A shortest u-v path is called a geodesic, and the diameter (denoted $d(G)$) of a connected graph G is the length of any longest geodesic.

A graph G is connected if every pair of points in $V(G)$ are joined by a path formed from members of $E(G)$. A maximal connected subgraph of G is called a component of G. A cutpoint of a graph is one whose removal increases the number of components and a bridge is such a line. The connectivity $\chi = \chi(G)$ of a graph G is the minimum number of points whose removal results in a disconnected graph. A graph G is n-connected if $\chi(G) \geq n$.

A graph G is planar if it can be drawn on the plane so that no two edges intersect. A tree is a connected graph with no cycles.

A complete graph K_p has every pair of p points adjacent. A clique of a graph is a maximal complete subgraph.

The following sequence of definitions refer to types of graphs which are the source of greater difficulty to existing isomorphism algorithms. The strongly-regular graphs in particular cause a higher order of computation from Corneil's algorithm (see section 1.5.1) and require many backtracking branches from Druffel's algorithm (see 1.5.4). The smallest known counter-example to Corneil's automorphism conjecture is a 25-point graph of a design (discovered by Mathon). However, an algorithm by Mathon (see section 1.5.4) is capable of handling many graphs of designs very well. As will be observed in later chapters, certain graphs of designs and strongly-regular graphs cause some backtracking in the algorithm of this thesis in order to determine isomorphism or to find the generators of the automorphism group of a graph.

A graph G which is not complete and not void, is 2-strongly regular (from Corneil [1968]) if there exists constants $p_{11}^1, p_{12}^1, p_{22}^1, p_{11}^2, p_{12}^2, p_{22}^2$ where:

I for $\forall y \in V(G)$, $\forall z \in V(G)$, where $(y,z) \in E(G)$

(i) $|\{x | x \in V(G);\ (x,y) \in E(G);\ (x,z) \in E(G)\}| = p_{11}^1$

(ii) $|\{x | x \in V(G);\ (x,y) \in E(G);\ (x,z) \notin E(G)\}| = p_{12}^1$

(iii) $|\{x | x \in V(G);\ (x,y) \notin E(G);\ (x,z) \notin E(G)\}| = p_{22}^1$

II for $\forall y \in V(G)$, $\forall z \in V(G)$, where $(y,z) \notin E(G)$

(i) $|\{x | x \in V(G);\ (x,y) \in E(G);\ (x,z) \in E(G)\}| = p_{11}^2$

(ii) $|\{x | x \in V(G);\ (x,y) \in E(G);\ (x,z) \notin E(G)\}| = p_{12}^2$

(iii) $|\{x | x \in V(G);\ (x,y) \notin E(G);\ (x,z) \notin E(G)\}| = p_{22}^2$

Note that this definition may be extended to define an h-strongly regular graph (where the upper bound for h is n). A more tangible definition of 2-strongly regular is stated by Bose [1963]: a regular graph is said to be 2-strongly regular if (i) any two vertices which are joined in G, are both simultaneously joined to exactly p_{11}^1 other vertices, and (ii) any two pairs of vertices which are unjoined in G, are both simultaneously joined to exactly p_{11}^2 vertices.

Let X be a v-set of objects or treatments $\{x_1, x_2, \ldots, x_v\}$ and let $X_1, X_2, \ldots, X_b$ be b distinct subsets of X. These subsets are called a balanced incomplete block design (BIBD) or a (b,v,r,k,λ)-configuration provided they satisfy the following requirements:

(i) each X_i is a k-subset of X.

(ii) each 2-subset of X is a subset of exactly λ of the sets $X_1, X_2, \ldots, X_b$

(iii) the integers v, k, and λ satisfy $0 < \lambda$ and $k < v-1$

(iv) any $x \in X$ occurs in exactly r of the subsets $X_1, X_2, \ldots, X_b$

Suppose we have a BIBD on v objects $\{x_1, x_2, \ldots, x_v\}$ and b distinct subsets of the objects, $X_1, X_2, \ldots, X_b$. Then by a graph of this design we mean a bipartite graph:

(i) which has v+b nodes where the first v nodes represent the objects, and the next b nodes represent the subsets.

(ii) in the adjacency matrix, $a_{ij} = 0$ for $\forall i,j \in \{1,2,\ldots,v\}$ and $a_{ij} = 0$ for $\forall i,j \in \{v+1,\ldots,v+b\}$.

(iii) for $\forall i \in \{1,2,\ldots,v\}$ and for $\forall j \in \{v+1,\ldots,v+b\}$ $a_{ij} = 1$ if object x_i is contained in subset X_{j-v} in the given design, otherwise $a_{ij} = 0$.

(iv) similarily, for $\forall i \in \{1,2,\ldots,v\}$ and for $\forall j \in \{v+1,\ldots,v+b\}$ $a_{ji} = 1$ if subset X_{j-v} contains object x_i, otherwise $a_{ji} = 0$.

1.2 Evaluating an Isomorphism Algorithm

A necessary part in the course of presenting a new algorithm is an evaluation of its efficiency (both time and storage) in order to compare its performance with other algorithms. Recently there has been a great surge of graph algorithms each trying to "outdo" the others in terms of computational efficiency. D. Corneil [1974 & 1975] has compiled a comprehensive survey of the development and analysis of many graph algorithms. Almost without exception Graph Theorists and Computer Scientists have adopted the techniques and conventions from the Theory of Computational Complexity in order to measure this entity of efficiency in consideration of combinatorial algorithms. D. Kirkpatrick [1974] discusses a number of these techniques in his Ph.D. thesis.

The concept of a polynomial-bounded algorithm provides a theoretical estimate (often pessimistic) of the worst performance of a given algorithm. The commonly accepted notion of polynomial-bounded algorithm (for graphs) is one which:

(i) guarantees a correct solution

(ii) can be executed in time T (even in the worst case) on an imaginary[1] computer of conventional design, where T is bounded by a polynomial usually expressed in n (the number of vertices) or m (the number of edges) or in terms of both n and m.

Various difficult combinatorial problems for which no polynomial-bounded algorithms are known, and which are equivalent from a complexity point of view, have been grouped together and called "NP-complete". Cook [1970] and Karp [1972] have shown that all NP-complete problems can be solved in polynomial time on a non-deterministic Turing machine, and further-more, if any one NP-complete problem can be solved in polynomial time on a one-tape deterministic Turing machine, then all NP-complete problems can be solved in polynomial time on such a machine.

The subgraph isomorphism problem is NP-complete, but the graph isomorphism problem is not known to be NP-complete. There are some polynomial isomorphism algorithms known for specific classes of graphs but there is not a general one for all classes of graphs. For example, Hopcroft and Tarjan [1972] have developed an O(n) isomorphism algorithm for trees. More recently the problem has perhaps been put into proper perspective with K. Booth's [1976] proof that graph isomorphism is equivalent to regular graph isomorphism. In view of this it is somewhat disheartening (but perhaps not surprising) that regular graphs (and

[1]For this purpose, the two most commonly used "imaginary" computers are the Turing machine (see Turing [1936]) and Cook's [1972 & 1973] model of a Random Access Machine (or RAM).

especially strongly regular graphs) are a source of difficulty to most algorithms.

One of the best general isomorphism algorithms so far, is due to Corneil [1968 & 1970]. The computation time of his algorithm depends on n^{5+h} (where $2 \leq h \leq n$ and h relates to the strong regularity of the graphs or their subgraphs). So for graphs that do not have a 2-strongly regular subgraph, his algorithm is considered efficient. However, Corneil's algorithm depends upon a conjecture (which has since been disproven) and so it becomes partly a backtracking algorithm which effectively handles a large class of graphs. The smallest counter-example to Corneil's conjecture is a 25-point graph of a design (found by R. Mathon and reported in Corneil [1974]). More recently two effective backtracking algorithms have been developed which handle a large class of graphs (including some graphs of designs): one is from Mathon [1975] and the other is from Druffel [1975 and 1976]. In addition B. McKay [1976] has presented an algorithm which handles strongly regular graphs. All four of these algorithms will be discussed in more detail in section 1.5.

A majority of the isomorphism algorithms are heuristic and thus use backtracking which cannot be evaluated on an "imaginary" computer. Instead they have been evaluated via a statistical study of their execution on a real computer. In the case of graph isomorphism we claim that it is more meaningful to measure performance on specific known classes of hard examples rather than on randomly generated graphs. Simple graph properties will often distinguish between randomly generated graphs whereas a set of strongly regular graphs (for example) might demonstrate the power of a given algorithm. At best a statistical study can predict an average performance of the algorithm; as it is impossible to guarantee that a "worst case" has been processed. (Knuth [1975] discusses evaluation of backtrack programs.) Lastly, if possible it is important to establish

how large a class of graphs an algorithm will work for (in particular the ones it will not handle successfully and why). In the rest of this chapter we review various algorithms noting their computation time and the classes of graphs for which they are successful.

1.3 Early Methods of Solving the Problem

For graphs of order n whose points are labeled with the same set of elements, there is an inefficient deterministic isomorphism algorithm called the Node Reordering Algorithm which is based upon S_n, the set of all n! permutations on n points. Each element of S_n is used to permute the points of a graph G, and then a test is made to see if this is identical to a given graph H. If the two graphs are identical then the process terminates. If the two graphs are not identical then the next element of S_n is used. This algorithm always yields a solution; however, the number of iterations for a given graph is bounded by n!. Hence the algorithm is not acceptable because the computation time and memory requirements are quite excessive.

Pólya's theory of enumeration (see Harary [1969] and Pólya [1937]) allows one to count the total number of graphs on n points, but it is not clear how to generate these graphs directly. Pólya's theory views the set of graphs on n points as a set of functions f_i mapping D (the set of n(n-1)/2 unordered pairs of distinct integers from $\{1,2,\ldots,n\}$) onto the set R = $\{0,1\}$. For n = 4 and D = $\{12,13,14,23,24,34\}$ the functions are of the form:

$$f_k: \begin{array}{cccccc} 12 & 13 & 14 & 23 & 24 & 34 \\ \hline \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 \end{array} \quad \text{where } \alpha_i \in \{0,1\}.$$

If ij in D is a line of G then $f_k(ij)=1$; otherwise $f_k(ij)=0$. Thus all possible graphs on n points may be represented as $\{f \mid f:D \rightarrow R\}$.

Note that any permutation σ in S_n induces a permutation σ' on D. Thus, using Pólya's notion of equivalence, two functions (graphs) f_1 and f_2 are equivalent (isomorphic) if there exists a permutation $\sigma \epsilon S_n$ such that $f_1(d)=f_2(\sigma'(d))$ for all d in D, where σ' is the permutation on D induced by σ. Although Pólya's theory uses this principle (it is in fact a realization of the Node Reordering Algorithm) to enumerate the number of graphs on n points, no one has been successful in finding an algebraic method of directly dividing this set of functions into the desired equivalence classes to obtain the representative non-isomorphic graphs. (Harary [1955] first used this technique to count graphs.)

In an effort to reduce the large problem space of n! possible solutions, people began to utilize various graph properties. Foremost among these, is the degree sequence of the graph. It is obvious that two graphs cannot be isomorphic if they do not have the same degree sequence. Furthermore, for two graphs G and H of the same degree sequence, a node of G could be mapped only onto those points of H of the same degree. In this way, the number of possibilities in the reordering is greatly reduced.

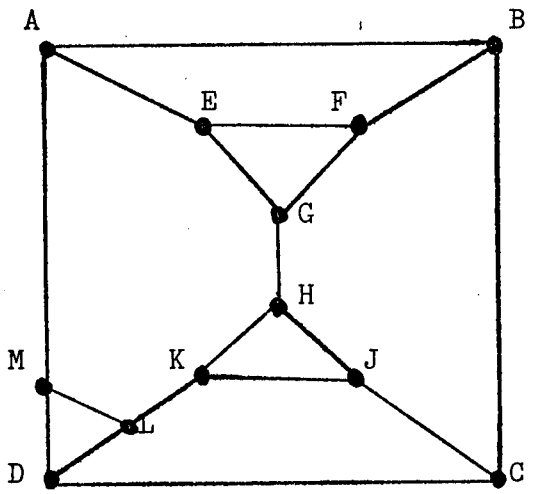
The degree sequence property does not reduce the number of reorderings for regular graphs and so it becomes necessary to look for more discriminating properties. Toward this end, R. Frucht [1949] introduced the notion of 'type' of a vertex, which was based on the cycle structure of the graph. According to his scheme, only vertices of the same 'type' may be mapped onto one another.

Frucht defines 'type' in the following way:

Let vertex v of a graph G have degree d. Let $p = \{p_1, p_2, \ldots, p_d\}$ be the set of all points in V(G) that are adjacent to v. Consider all possible unordered pairs of distinct elements from the set P where $P = \{p_1p_2, p_1p_3, \ldots\ldots, p_2p_3, \ldots\ldots, p_{d-1}p_d\}$. Let the set $U = \{u_1, u_2, \ldots, u_s\}$ be in 1-1 correspondence with the set P. For each pair p_ip_j in P, set the corresponding u_k equal to the size of the smallest cycle containing

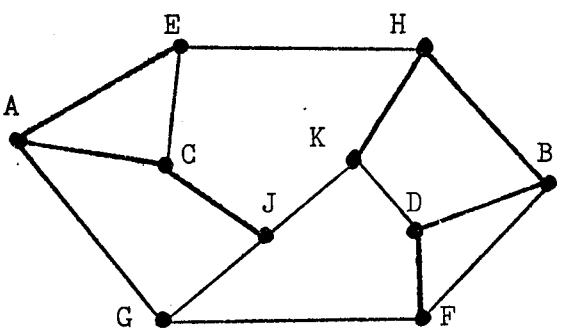
both edges vp_i and vp_j. If there is no cycle containing those edges then set $u_k = \infty$. Lexicographically order U so that $u_1 \leq u_2 \leq \ldots\ldots \leq u_s$. Then the s-tuple $(u_1, u_2, \ldots\ldots, u_s)$ is called the type of vertex v.

Consider the two cubic graphs and their node types shown in Figures 1.3.1 and 1.3.2:



Nodes	Type
A	(4,5,7)
B	(4,5,6)
C	(5,5,6)
D	(3,5,5)
E,F	(3,4,5)
G,H	(3,6,7)
J,K,L,M	(3,5,6)

Figure 1.3.1 A 12-point cubic graph and its node types.



Nodes	Type
A,B,C,D	(3,4,5)
E,F	(3,5,6)
G,H	(4,5,6)
J,K	(4,5,5)

Figure 1.3.2 A 10-point cubic graph and its node types.

Compared with the full Node Reordering, the computation time is reduced from 12! to 2!x2!x4! = 96 for the graph in Figure 1.3.1 and from 10! to 4!x2!x2!x2! = 192 for the graph in Figure 1.3.2. Although it is clear that this heuristic is effective in reducing the problem space, the computation time to determine the type is still considerable. However, this method becomes more practical as better cycle finding algorithms are developed. The only other drawback is that there exist graphs in which there are a large number of nodes with the same type. In a later section we will discuss some results due to Turner [1967 & 1968] which essentially point out that the cycle structure of a graph is not sufficient to discriminate between non-isomorphic graphs.

A number of different people have interpreted isomorphism as a coding problem. See for example, Nagle [1966], Randic [1974], Shah et al. [1974], and Stockton [1968]. The chief aim of their approach is to find a code which represents a graph such that two graphs are isomorphic if and only if they have the same code. The more or less common approach is to consider all possible ways of rewriting the upper triangle (for undirected graphs) of the adjacency matrix in order to discover the largest binary integer which is formed by concatenating the rows of the adjacency matrix above the diagonal. For all intents and purposes this method is equivalent to the Node Reordering Algorithm and hence is just as impractical. Corneil and Read [1975] comment that this method could be improved by first of all subdividing the vertices into classes (according to some criterion) then the number of labellings is markedly reduced. They further comment that another major disadvantage is that the code cannot be determined until all the labellings (for classes) are found. They suggest that perhaps a breadth-first or a depth-first search might be used to reduce the work.

1.4 Heuristics For Determining Isomorphism on the Computer

In order to generate graphs on the computer it is necessary to contend with graph isomorphism. This is the critical part of the generation algorithm because once a new graph is generated we must be able to recognize if it is indeed a new graph. It is worthwhile noting here that except for Frucht, the methods of both this section and the previous one fail to deal very satisfactorily with the regular graphs or a subset of them. At least Frucht's method can distinguish between some regular graphs (ie. those whose cycle structure is dissimilar).

S. H. Unger [1964] was one of the first persons to adopt a heuristic approach for finding graph isomorphisms of directed graphs on a computer. The major function of his program GIT (Graph Isomorphism Tester) is to find properties of the nodes that can be used to reduce the number of possible transformations between the given two graphs. Briefly, some of the nodal properties he uses are:

(1) in-degree and out-degree of the nodes

(2) the number of nodes that can be reached from node i along a path of length n

(3) the number of nodes that reach node i along a path of length n

(4) a binary-valued function that is equal to 1 on those nodes which are included in circuits of length n.

Unger's program either finds the isomorphism, disproves isomorphism, or comes to no conclusion if the graphs are regular with respect to his nodal functions. Sussenguth [1965] uses similar tactics to partition the nodes for determining isomorphism of chemical structures (he has the further advantage of being able to apply non-graphical properties if they are appropriate).

A. T. Berztiss [1973] developed a backtracking procedure for determining isomorphism of directed graphs based on a representation by linear formulas introduced by Krider [1964]. The notation is based on the representation of a directed arc by a binary-prefix operator * (called a K-operator) applied to the node names: an arc or directed edge $\langle a,b \rangle$ is represented by *ab (a K-formula). The algorithm generates a minimal set of K-formulas that represent a given digraph and uses this to form a refererence K-formula for one of the digraphs. It then selects all K-formulas representing the other digraph that have the same pattern. Each such correspondence defines an isomorphism. (Note that this is done in stages by comparing K-formulas of substructures.) During the matching procedures, large classes of permutations are eliminated from further consideration when it is found that certain groups of symbols cannot match with one another.

As Berztiss points out, it is difficult to establish the time dependence of a backtrack procedure because "the theoretical timing analysis of a backtrack procedure is close to a contradiction in terms". In view of this, Berztiss experimentally studied the timings for sets of 25 randomly generated pairs of non-isomorphic digraphs on n nodes for $n = 6, 8, 10, 12$ in which each node had its indegree and outdegree equal to $\frac{1}{2}n$ and there were no loops. He found that the timings were not efficient in the sense of polynomial bounds. He reports that some of the mean times could be fitted reasonably well by $t_n = 2.15 \times 10^{-4} \times (\exp(1.07n))$ but the ratios t_n/t_{n-2} were not constant, and so this would indicate a less favourable time dependence.

Baker, Dewdney, and Szilard [1974] generated all the unlabelled 9-point graphs on a computer. This happens to be the largest known complete set of graphs. Their program which is written in MACRO-10

assembly language, was run on a 165-K (words) PDP-10 and it generated the complete set of 274,868 graphs in less than 6 hours of CPU time. The graphs are generated iteratively in cases by the number of lines q, from q = 1 until 18, as the other 9-point graphs up to q = 36 lines are the complements of these graphs. The set of 9-point graphs with i lines is obtained from the set with i-1 lines by adding one line in all possible ways and then eliminating or keeping the new graph by using an isomorphism tester to compare it with a growing list of the ones already generated with i lines. Their isomorphism method is the obvious one of permuting the points in all possible ways. However, they use a heuristic to greatly reduce the number of possible permutations. Briefly, they use the degree of a node and the number of 3-cycles it is incident with to "type" the nodes and hence restrict the possible mappings. It is apparent that their heuristic is effective for graphs in general, but it is considerably less efficient for the case of regular graphs.

1.5 More Sophisticated Isomorphism Methods

In this section further isomorphism algorithms will be discussed in more detail. Some of these will serve as a basis for comparison in a later chapter.

1.5.1 Corneil's Algorithm (1968)

Corneil's algorithm [1968 & 1970] was the first efficient procedure for deriving the isomorphism partitioning for a general class of regular graphs. We will give a brief overview of the general tactics of his somewhat elaborate algorithm.

For any given graph G, the algorithm defines two graphs: the representative graph (denoted G_R) and the reordered graph (denoted G_r). The representative graph is constructed in such a way that it is a homomorphic image of the original graph G with uniquely labelled vertices. Corneil shows that the time required to determine the representative graph depends on n^{5+h}, where n is the number of vertices of G and h indicates the strong-regularity of the graph or its subgraphs (see section 1.1 for the definition of 2-strongly regular). Hence for graphs that do not contain an h-strongly regular subgraph, the computation time is polynomial in n^5. Two representative graphs are isomorphic only if they are identical because of the unique labelling of the nodes. Thus, if the representative graphs of two given graphs are not identical, then the given graphs are not isomorphic. Corneil uses the converse of this to conjecture that if the representative graphs are identical then the given graphs are isomorphic.

The reordered graph G_r, that is constructed from the representative graph G_R to be isomorphic to the given graph G, is used to verify the isomorphism. It is shown that if the reordered graphs are identical then the given graphs are isomorphic. The computation time to determine the reordered graph G_r depends upon n^{5+h}.

The algorithm terminates with one of the following statements:

(i) the graphs are isomorphic

(ii) the graphs are not isomorphic

(iii) the graphs form a counter-example to the conjecture.

Corneil proved his conjecture for trees and failed to find a counter-example for a general graph over a large number of test cases. However, he reports (see Corneil [1974] p.26) that in 1973, R. Mathon discovered some counter-examples in the area of isomorphism of balanced incomplete

block designs and that the smallest counter-example is of order 25. With the advent of such a counter-example, the algorithm is reclassified as backtracking, but still remains effective over a large class of graphs.

Levi [1974] reports that Sirovich [1971] developed a similar algorithm to Corneil's; however, in addition Sirovich gives a procedure for deriving all the isomorphisms between two isomorphic graphs. Although the computation time of Sirovich's algorithm is $O(n^5)$ for all graphs, he does not prove the derivation of his isomorphism partitioning.

1.5.2 Saucier's Algorithm (1971)

G. Saucier [1971] has developed an isomorphism algorithm which constructs a partitioning of the vertices of the graph. Although the method is not based on conjecture, it is difficult to evaluate as Saucier does not present a very detailed complexity analysis. We will comment further on the efficiency after the following brief explanation by example of the principal ideas in her algorithm.

Given a graph $G=(V,E)$, the first step is to partition V into a number of classes $\{C_0,.....,C_i,.....\}$ by the following procedure:

(i) choose a vertex $S \in V$

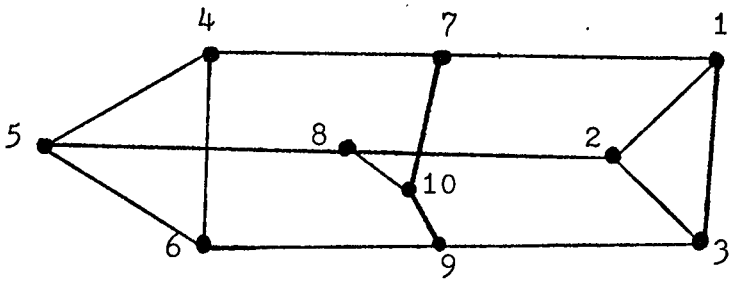
(ii) put $S \in C_0$

(iii) for $\forall x \in V$, $x \in C_i$ iff the shortest distance between x and S is i.

The next step is to partition the nodes in each class C_i into sub-classes $\{c_0,c_1,.....\}$ by the following method:

To each point y in a C_i, where $i \geq 1$, assign an ordered triple (α, β, γ) where α is the number of nodes in C_{i-1} adjacent to y, β is the number of nodes in C_i adjacent to y, and γ is the number of nodes in C_{i+1} adjacent to y.

As an illustration of the procedure so far, observe the classes and sub-classes of the 10-point cubic graph in Figure 1.5.2 with node 5 chosen as S.



C_0	C_1	C_2	C_3
5 } c_0	4 (1,1,1) } c_0 6 (1,1,1) } c_0 8 (1,0,2) } c_1	7 (1,1,1) } c_0 9 (1,1,1) } c_0 10 (1,2,0) } c_1 2 (1,0,2) } c_2	1 (2,1,0) } c_0 3 (2,1,0) } c_0

Figure 1.5.2 Saucier's initial parition of a 10-point graph.

The final step in Saucier's method is to refine if possible the sub-classes within each class C_i by assigning to each node y, a triplet of series (m_1, m_2, m_3) as follows:

(1) m_1 is a series formed by the concatenation of the sub-class names of the points in C_{i-1} that are adjacent to y.

(2) m_2 is likewise a concatenation of the sub-class names of the points in C_i adjacent to y.

(3) m_3 is formed in the same fashion with the points of C_{i+1}.

Now after each point in a class C_i is assigned a triple (m_1, m_2, m_3), the points can be divided into further sub-classes and arranged into lexicographical order according to whether or not they have the same triple. For example in Figure 1.5.2 nodes 4 and 6 are both assigned the

triple (c_0,c_0,c_0) and so there is no further refinement.

In summary, given two graphs G and G' on n nodes, the method would:

(1) select an $S \in V(G)$

(2) determine the classes $\{C_0,C_1,.....\}$

(3) refine each class initially through the use of the (α,β,γ) triple

(4) repeatedly refine the sub-classes of C_i using the (m_1,m_2,m_3) triples until no further refinement is possible

(5) select an $S' \in V(G')$, if all possible choices for S' are exhausted then stop

(6) determine the classes $\{C'_0,C'_1,.....\}$

(7) if the number of elements and the initial groupings within each class do not match between C_j and C'_j for $\forall j$ then go back to step 5 and select a different S'

(8) refine each C'_k and check for correspondence with C_k for $\forall k$. If they do not correspond, go to step 5 and choose a new S'.

(9) check the possible mappings for isomorphism. If there is no isomorphism then go back to step 5 and choose a new S'.

As Saucier points out, in the worst case there are (n+1) performances of the refinement algorithm: once for G and a maximum of n times for G' since it has only n points. However, the refinement algorithm does not guarantee singleton classes of nodes, and so the number of possible mappings to be tested for isomorphism after refinement may be quite large, especially if the graph is highly regular.

1.5.3 Levi's Algorithm (1974)

Levi [1974] proposed a new algorithm for testing isomorphism of directed and undirected graphs. His method was revolutionary in that he made use of several partitioning rules which are based on both node

and edge properties; whereas previous algorithms partitioned the node sets only. He presents an $O(n^5)$ classification procedure for partitioning the node and edge sets as far as possible, plus a heuristic procedure for deriving all the isomorphisms. The heuristic procedure examines a representation of the partition as a connectivity graph on which a sufficient condition for isomorphism can be tested.

Basically, the method uses a node feature f and an edge feature h to refine an initial partition of the sets of nodes and edges into equivalence classes (where a feature is considered to be an isomorphism invariant property of a node or an edge). The refinement proceeds until either non-isomorphism is detected or one of the edge or node sets cannot be refined any further.

As an example consider the two isomorphic graphs shown in Figure 1.5.3.1:

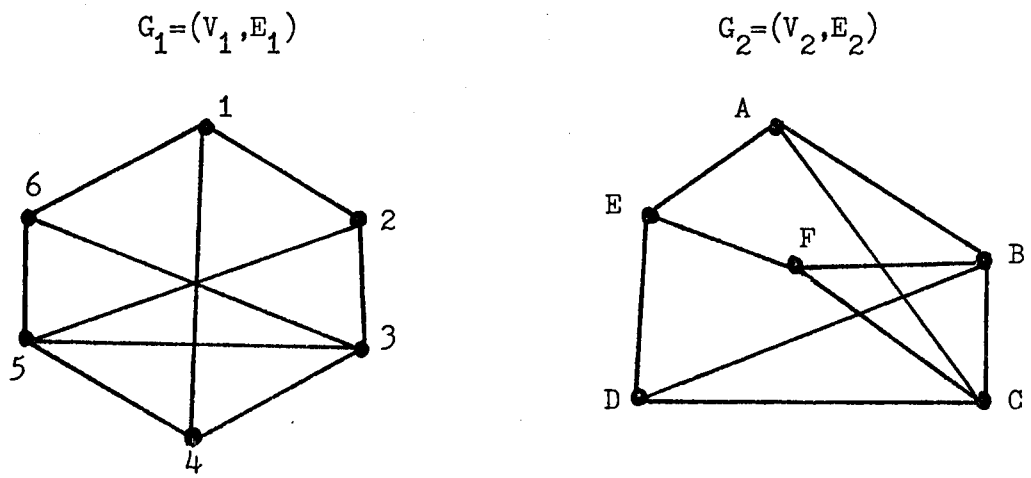


Figure 1.5.3.1: Two 6-point isomorphic graphs.

With S_0 and T_0 as the initial node and edge partitions respectively, the refinement proceeds as follows:

$S_0 = \{(\{3,5\},\{B,C\}), (\{1,2,4,6\},\{A,D,E,F\})\}$

$T_0 = \{(\{E_1\},\{E_2\})\}$

$S_1 = S_0$

$T_1 = \{(\{12,14,16\},\{AE,DE,EF\}),\ (\{35\},\{BC\}),\ (\{23,25,34,36,45,56\},$
$\{AB,AC,BD,BF,CD,CF\})\}$

$S_2 = \{(\{3,5\},\{B,C\}),\ (\{1\},\{E\}),\ (\{2,4,6\},\{A,D,F\})\}$

$T_2 = T_1$

Observe that the initial partition in this example is based on node degree. In the second step T_0 is refined as follows: the first two pairs of equivalent classes of edges are all those edges which join equivalent nodes from S_0 (that are also equivalent in T_0). In the third step nodes 1 and E split off from their previous class in S_0 because they do not occur in an edge of the third pair in T_1 as the rest of their previous class members do. The refinement ends here.

For the next step, the above partitions of the node and edge sets are represented by a weighted undirected graph which Levi calls the connectivity graph. The number of nodes in the connectivity graph is equal to the number of pairs of equivalent nodes in the final node partition set S_k. In our example there are three such pairs in S_2. Each node of the connectivity graph has an associated weight which is equal to the number of equivalent nodes in the partition pair to which the node corresponds. If v_1 represents the first pair in S_2 then v_1 has weight 2 (denoted $v_1(2)$). The edges of the connectivity graph are weighted and each edge corresponds to a partition pair from the final edge partition set. The weight of each edge u_i is equal to the number of equivalent edges in the corresponding partition pair. A given edge u_i joins nodes v_j and v_k if the edges of the corresponding edge partition pair join equivalent nodes from the node partition pair represented by v_j to equivalent nodes from the node partition pair represented by v_k. In our example the first edge u_1 has weight 3 and it

joins nodes $v_2(1)$ and $v_3(3)$. The whole connectivity graph for our example is given in Figure 1.5.3.2:

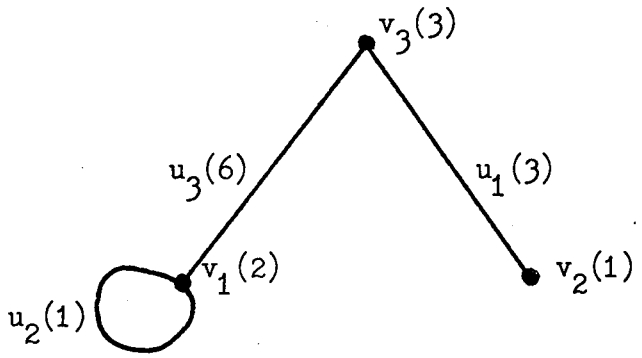


Figure 1.5.3.2 Levi's connectivity graph for the node and edge partitions of the examples in this section.

Although he presents no general efficient procedure to decide if a connectivity graph corresponds to the isomorphism partitioning, Levi gives a heuristic algorithm which exploits properties of the connectivity graph in trying to derive the isomorphisms. (The algorithm resorts to other features and extended features if necessary in attempting to further subdivide the node and edge sets.) For the heuristic algorithm Levi claims an average number of n^6 operations required to derive one isomorphism for most graphs. This would appear to be a relatively good time, but it is difficult to evaluate as he gives no information on any particular worst cases.

1.5.4 Algorithms of Druffel (1975) and Mathon (1975)

Both Druffel [1975 & 1976] and Mathon (see in Gibbons [1976]) have developed fast backtracking isomorphism algorithms which are capable of dealing with strongly regular graphs. In addition Mathon's algorithm can successfully deal with the 25-point graphs of a design which form a counter-example to Corneil's conjecture (his automorphism partitioning in Corneil [1968]) as well as some other designs. P. Gibbons [1976]

has adapted the basic techniques of both these algorithms to produce a backtracking algorithm which applies specifically to designs. Since these are probably the best isomorphism algorithms so far, we will briefly describe their tactics and measured results in order that we may refer later to them for comparison.

Mathon's algorithm takes into account the number of cliques up to a given constant size (and also void subgraphs if necessary) that contain a given vertex. For a design D, this analysis is carried out on the block intersection graph G_i (i=0,1,2,....,k where k is the size of the subsets) which is defined as follows:
The vertices of G_i represent the blocks of D, and two vertices are adjacent iff the corresponding blocks contain exactly i treatments in common. By a (c,i)-clique analysis of a design D, Mathon means an analysis of the number of cliques of size c (where c is a constant) in the block intersection graph G_i of D. His backtrack procedure for producing a (c,i)-clique analysis outputs the results in the form of a matrix t(2::c,1::b), where t(i,j) is the number of i-cliques containing block j. Apparently this clique analysis provides a strong characterization for many classes of designs and is effective in distinguishing between some classes of strongly regular graphs. One drawback for really large n is that clique determination is known to be an NP-complete problem (see Aho et al. [1974]) and so the computation time is bound to grow enormously.

Druffel's algorithm takes advantage of the information contained in the distance matrix representation of a graph to establish an initial partition of the vertices of two directed graphs and then uses a backtracking procedure to reduce the search trees of possible mappings. The distance matrix D is an nxn matrix in which the element $d_{i,j}$ represents the length of the shortest path between the vertices v_i and v_j (if i=j

then $d_{i,j}=0$). Druffel uses Floyd's [1962] $O(n^3)$ algorithm to find the distance matrix. Then he forms a characteristic matrix by concatenation of row characteristic and column characteristic matrices as follows: a row characteristic matrix XR is defined to be an nx(n-1) matrix such that element $xr_{i,m}$ is the number of vertices which are a distance m away from v_i. Similarily a column characteristic matrix XC is formed by the concatenation of the corresponding rows of XR and XC. Vertices with identical rows in the characteristic matrix are assigned to the same class. This initial partition requires $O(n^3)$ time since the distance matrix requires $O(n^3)$ and the realization from there requires $O(n^2)$.

The backtracking algorithm selects possible vertex mappings between the two graphs and then checks these for consistency by examining the consequences on the respective distance matrices. If it is not possible to find a consistent mapping then it is necessary to backtrack and try another mapping. The lower bound for this algorithm is $O(n^2)$ and the somewhat pessimistic upper bound is $O(n.n!)$. In order to improve this, Druffel uses a dynamic bound for which he analyzes the backtracking for three different strategies, each of which shape the backtrack tree by making heuristically determined choices.

The algorithm was tested on randomly generated regular graphs in which case the graphs were either rejected as non-isomorphic at the initial partition or they required no backtracking to prove isomorphism. The following Table summarizes the predicted dynamic bound and actual number of branches for all three backtracking strategies during the comparison of fifteen non-isomorphic strongly regular graphs of degree 12:

DYNAMIC BOUND AND ACTUAL NUMBER OF BRANCHES
FOR STRONGLY REGULAR GRAPHS OF ORDER 25

G_1-G_2	Strategy 1		Strategy 2		Strategy 3	
	Bound	Run	Bound	Run	Bound	Run
S_1-S_2	37825	4129	39825	5773	25825	5023
S_1-S_3	37825	4129	19825	5893	25825	5083
S_1-S_5	37825	9385	19825	8017	25825	7081
S_1-S_8	37825	4063	19825	5585	25825	4861
S_1-S_{14}	37825	4059	19825	5565	25825	4849
S_2-S_1	97825	1825	15325	1825	61825	1825
S_3-S_1	97825	1825	15325	1825	97825	1825
S_4-S_6	7825	1825	76825	1825	54325	1825
S_5-S_7	97825	1825	19825	4561	13825	3985
S_6-S_4	115825	1825	15325	3847	61825	1825
S_8-S_9	97825	1825	15325	4781	97825	1825
S_8-S_{10}	97825	2003	15325	5153	97825	1959
S_9-S_1	7825	1825	54325	1825	39325	1825
S_{10}-S_{11}	97825	2329	15325	5221	97825	2239
S_{13}-S_1	97825	1825	15325	1825	79825	1825
S_{15}-S_1	97825	1825	15325	1825	61825	1825

Table 1.5.4 Performance of Druffel's algorithm on some 25-point strongly regular graphs.

Overall the three strategies seem somewhat comparable and most certainly required far less than the predicted bound. We will refer back to this Table for comparison in a later chapter.

1.5.5 McKay's Algorithm (1976)

More recently B. McKay [1976] has developed an algorithm for canonically labelling a graph through the use of a theory of backtrack programming and of the invariance group of such a program. The method determines elements of the automorphism group Γ_G during the labelling process in order to reduce the amount of work. It would appear that

the algorithm is very efficient as McKay claims that it finds a set of no more than $|V(G)| - p$ generators for Γ_G where Γ_G has p orbits. For large random graphs he claims that it is impossible to devise an algorithm which is very much faster.

1.6 Isomorphism of Planar Graphs

It is worthwhile noting that recent research on the isomorphism of planar graphs has been very successful. Weinberg [1966] exploited the fact that a triconnected planar graph has a unique embedding on a sphere, to develop an algorithm for testing isomorphism of triconnected planar graphs in $O(|V|^2)$ time where V is the set consisting of the vertices of both graphs. Hopcroft and Tarjan [1971, 1972, & 1973] extended this result to arbitrary planar graphs and improved it to $O(|V|\log|V|)$. Lastly Hopcroft and Wong [1974] improved the algorithm to a linear time bound of $O(|V|)$. However, they report that although their algorithm has a linear asymptotic growth rate, it appears to be inefficient due to the presence of a rather large constant. In view of this, they feel that their result is important in that it establishes the existence of a linear algorithm which subsequent work might improve upon and provide a practical linear algorithm.

1.6.1 Isomorphism of Trees

Since trees are a subset of planar graphs the above mentioned algorithms include trees as well. So Hopcroft and Tarjan [1972] have produced an algorithm of $O(|V|)$ for trees. In addition, Corneil [1968] proves his isomorphism conjecture for the case of trees; so he provides an efficient n^5 algorithm for testing isomorphism of trees.

C. Snow [1973] has produced some interesting results on the ordering of trees which may well provide a method of determining isomorphism of trees by direct comparison of the components of the trees. He establishes a method of ordering free trees, ordered rooted trees, and unordered rooted trees based on their representation by either height or weight vectors. Consider a tree $\mathcal{T}$ of the form given in Figure 1.6.1:

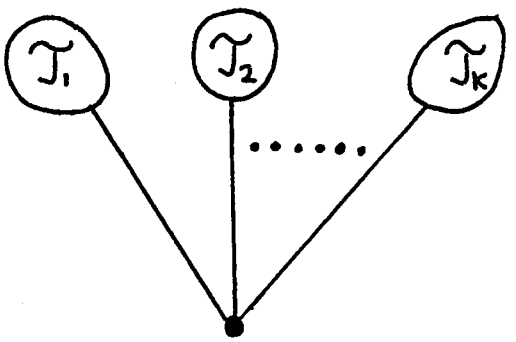


Figure 1.6.1 A tree in canonical form.

Basically he arranges $\mathcal{T}$ in a canonical form so that $\mathcal{T}_1 \geq \mathcal{T}_2 \geq \ldots \geq \mathcal{T}_k$ where each $\mathcal{T}_i$ is assumed to be in canonical form. Suppose there exists a tree $\mathcal{T}'$ in canonical form with sub-components $\mathcal{T}_1', \mathcal{T}_2', \ldots\ldots, \mathcal{T}_k'$ such that $\mathcal{T}_1' \geq \mathcal{T}_2' \geq \ldots\ldots \geq \mathcal{T}_k'$. Snow conjectures that one could determine whether or not $\mathcal{T}$ and $\mathcal{T}'$ are isomorphic by directly comparing whether or not the corresponding sub-components are identical.

1.7 The Search For Algebraic Isomorphism Invariants

Recently in the search for isomorphism invariants more attention has been directed to the algebraic properties of graphs. Various results about the eigenvalues, eigenvectors, and characteristic polynomial of the adjacency matrix are due to Hoffman [1963, 1969, & 1974], Harary [1962 & 1971], Izbicki [1960], Mowshowitz [1969, 1972, & 1973], Petersdorf and Sachs [1969], Wilf [1967], Bakhovskii [1965], Seidel [1968], Collatz and

Sinogowitz [1957], Doob [1970], Reischer and Simovici [1971], Böhm and Santolini [1964], and Cvetković [1971a & 1971b]. (Also see Biggs [1974] for a comprehensive survey of Algebraic Graph Theory.) In later chapters we will mention and make use of some of these results. At the moment, our attention is directed to several separate papers each of which investigate the use of matrix algebra to determine isomorphism.

J. Turner [1967 & 1968] attempted to define a more discriminatory set of matrix functions that would characterize graphs up to isomorphism. Since it had been shown that the characteristic polynomial is insufficient to distinguish between non-isomorphic graphs, he decided to generalize the idea of the characteristic polynomial through the use of generalized matrix functions. His main results relate the generalized characteristic polynomial with the geometrical properties of the graph.

He defines a dissection of a graph G to be a collection of paths in G having the following properties:

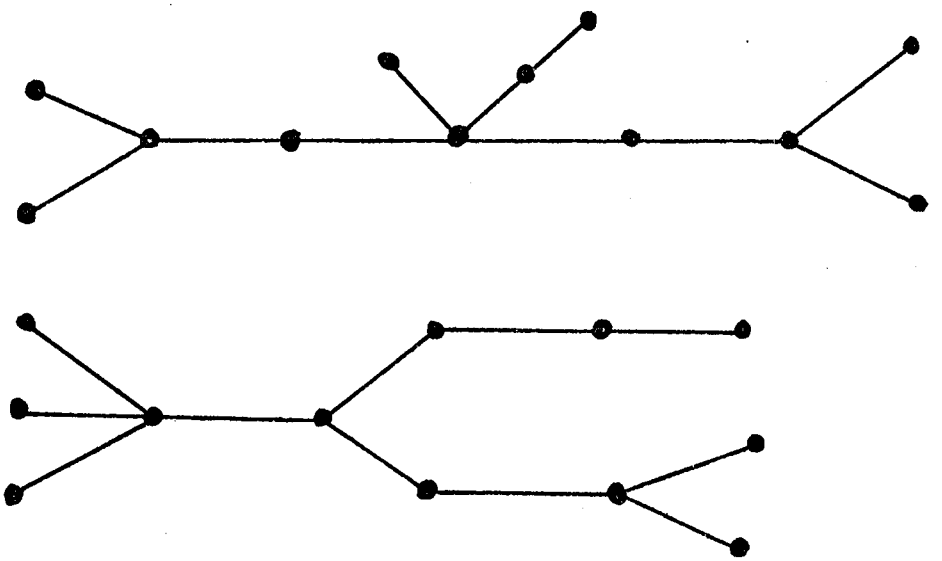
(1) each path is either a cycle, a single edge, or a single vertex

(2) every vertex of G lies precisely on one path.

Hence a dissection is associated with a certain partition of the integer n if the summands of the partition correspond to the number of vertices on the paths of the dissection. Turner's main result establishes that two n-point graphs G_1 and G_2 have the same generalized characteristic polynomial if they have the same number of dissections corresponding to each partition of the integer n.

Based on this result, Turner conjectured that two graphs are isomorphic if they have the same degree sequence and the same number of dissections corresponding to each partition of the integer n. Unfortunately Turner found a counter-example to his conjecture by searching for one on the computer. He discovered the two non-isomorphic trees (shown in Figure 1.7.1 with their number of dissections), which have the same

degree sequence and the same number of dissections. The significance of Turner's work is that he has demonstrated that the cycles structure of a graph (in the sense of numbers of different types of cycles) is not sufficient to discriminate between non-isomorphic graphs.



Partitions of n = 12	# Dissections
2+1+1+1+1+1+1+1+1+1+1	11
2+2+1+1+1+1+1+1+1+1	40
2+2+2+1+1+1+1+1+1	55
2+2+2+2+1+1+1+1	21
All others	0

Figure 1.7.1 Counter-example to Turner's conjecture.

A. Mowshowitz [1972] established some results addressed to the problem of determining under what conditions the characteristic polynomial of the adjacency matrix of a graph distinguishes between non-isomorphic graphs. He derives a formula for the coefficients of the characteristic polynomial of an arbitrary digraph expressed in terms of the number of collections of certain disjoint directed cycles. Based on this he proves the following theorem:

Theorem 1.7.1 (Mowshowitz [1972])

For any positive integer k there exists an integer n (the number of vertices) such that there are at least k non-isomorphic weakly connected digraphs with the same characteristic polynomial.

Mowshowitz extends his results and studies the polynomials of non-isomorphic cospectral trees in depth. Among these, he proves that there exist infinitely many pairs of non-isomorphic cospectral trees (and he gives the general form of these). This work formalizes the nature of the counter-examples to Turner's conjecture.

Reischer and Simovici [1971] have established some results on finding the solution to the matrix equation $PAP^T = B$ in the two-element Boolean Algebra. Given that A and B are 2-boolean matrices of the same order, they wish to find a permutation matrix P such that $PAP^T=B$. The first step in their method is to view A and B as adjacency matrices of directed graphs and from this to form a third directed graph C, which represents a correspondence between nodes of A and B that have the same in-degree and out-degree. Secondly, they prove that P is related to a factor of C. In effect what they have done is to partition the nodes of A and B into subsets of nodes that have the same in-degree and out-degree, and said that P is among the many possibilities of mapping the subsets of A onto their counterparts in B. Although the method does in fact find a solution if one exists, the node-partitioning is not sufficiently discriminating to reduce the number of possibilities significantly. In fact, in the case of regular graphs of order n, there would be n! possibilities.

Kuhn [1971a & 1971b] and Gelbart [1976] have both conducted an investigation of the use of eigenvalues and eigenvectors to determine graph isomorphism. Both works are based on the following observation: if A and B are the vertex adjacency matrices of two graphs, then the

graphs are isomorphic iff there exists a permutation matrix P such that $A=P^TBP$. Both algorithms deal successfully with graphs that have all simple eigenvalues; however, they fail to handle graphs with repeated eigenvalues efficiently (especially ones where all the eigenvalues are of high multiplicity). Kuhn suggests a method of repeatedly adding a vertex to the graphs in order to reduce the multiplicity of at least one eigenvalue by one each time a vertex is added. Gelbart uses an equation solving method to handle an eigenvalue of multiplicity k which in the worst case would require $O(\binom{n}{k}k^3)$ computation time. However in practice she uses backtracking and has found that the theoretical upper bound is far too pessimistic. For graphs whose eigenvalues are all simple she presents an $O(2^{\lceil n/2 \rceil})$ algorithm for finding the group of the graph. Here again she found that in practice the bound was far too high as her backtracking algorithm produces generators of the group and not necessarily all the automorphisms.

The intention of this thesis is to investigate and demonstrate how the eigenvalues and eigenvectors may be exploited to determine graph isomorphism efficiently, not only for graphs with simple eigenvalues, but in particular for graphs whose eigenvalues are of high multiplicity. In Chapter 2 we investigate how to detect the non-isomorphism of two given cospectral graphs (excluding certain harder cases) by inspection of the eigenvectors belonging to both simple and multiple eigenvalues. In Chapter 3 the algorithm incorporates backtracking to handle isomorphic graphs and the more difficult cases including strongly regular graphs and some graphs of designs. In Chapter 4 an efficient backtracking algorithm is given for determining the generators of the automorphism group of any graph (ie. regardless of the multiplicities of the eigenvalues). Lastly, Chapter 5 contains an evaluation and analysis of the algorithms along with some measurements of the performance of the backtracking for the more difficult cases.

CHAPTER 2

A Powerful Algebraic Filter

2.0 Algebraic Properties of a Graph

In this chapter we will investigate how the eigenvectors associated with the eigenvalues of the adjacency matrix may be used to distinguish between a pair of non-isomorphic cospectral graphs. As background knowledge to further development, a selection of results (some of which overlap) already known about the spectral properties of a graph are presented in this section. Most of these results are found in a comprehensive survey by Cvetković [1971 a & b] or in Biggs [1974]. For the proofs of these theorems see Birkhoff and MacLane [1953], Gantmacher [1960], Marcus and Minc [1964], or Seneta [1973]. The adjacency matrix is a non-negative symmetric (0,1)-matrix since the graphs are assumed to be undirected and without loops or multiple edges.

Theorem 2.0.1

The eigenvalues and eigenvectors of a real symmetric matrix are all real.

Theorem 2.0.2

The eigenvectors associated with distinct eigenvalues of a real symmetric matrix are mutually orthogonal.

Theorem 2.0.3

If λ is an eigenvalue of multiplicity $m(\lambda)$ of a real symmetric matrix, then there is a vector space of eigenvectors of dimension $m(\lambda)$ associated with λ.

A matrix A is called reducible if there is a permutation matrix P such that the matrix $P^T AP$ is of the form $\begin{bmatrix} X & 0 \\ Y & Z \end{bmatrix}$, where X and Z are square matrices. Otherwise, A is called irreducible. It is known

that the adjacency matrix of a graph G is irreducible iff G is connected. Hence, the following theorem due to Perron and Frobenius applies to connected graphs:

Theorem 2.0.4 (Perron-Frobenius)

An irreducible non-negative matrix A always has a positive eigenvalue r that is a simple root of the characteristic polynomial. The moduli of all the other eigenvalues do not exceed r. To the 'maximal' eigenvalue r, there corresponds an eigenvector $\underline{z}=(z_1,z_2,\ldots,z_n)$ of A with positive co-ordinates $z_i>0$ for $i=1,2,\ldots,n$.

Moreover, if A has h eigenvalues $\lambda_0=r,\lambda_1,\lambda_2,\ldots,\lambda_{h-1}$ of modulus r, then these numbers are all distinct and are roots of the equation: $\lambda^h-r^h=0$. More generally, the whole spectrum $\lambda_0,\lambda_1,\lambda_2,\ldots,\lambda_{n-1}$ of A, regarded as a system of points in the complex λ-plane, goes over into itself under a rotation of the plane by the angle $2\pi/h$. If $h>1$, then A can be put by means of a permutation into the following 'cyclic' form:

$$A=\begin{bmatrix} 0 & A_{12} & 0 & \ldots & 0 \\ 0 & 0 & A_{23} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & A_{h-1,h} \\ A_{h1} & 0 & 0 & \ldots & 0 \end{bmatrix}$$

where there are square blocks along the main diagonal.

Theorem 2.0.5

For the eigenvalues $\lambda_1,\lambda_2,\ldots,\lambda_n$ of the adjacency matrix of an undirected graph G without loops or multiple edges the following statements hold:

1. $\lambda_1,\lambda_2,\ldots,\lambda_n$ are real and $\sum_{i=1}^{n}\lambda_i=0$;
2. if G has no edges, then $\lambda_1=\lambda_2=\ldots=\lambda_n=0$;
3. if G has at least one edge, then the inequalities stated below hold for the greatest number $\lambda_1=r$ and for the smallest $\lambda_n=q$, from the spectrum:

   (i) $1\leq r\leq n-1$ and (ii) $-r\leq q\leq -1$.

In (i) the upper bound is reached iff G is a complete graph, while the

lower bound is reached iff G has as components, graphs $K_{1,1}$[1] or isolated vertices, where at least one $K_{1,1}$ must exist. In (ii) the upper bound is reached if G contains complete graphs as components, and the lower bound is reached iff the component of G having the greatest eigenvalue, is a bipartite graph. If G is a connected graph, the lower bound in (i) is replaced with $2\cos\frac{\pi}{n+1}$. Then equality holds iff G is a tree with two vertices of degree one.

Theorem 2.0.6

Let A be the adjacency matrix of a regular graph G of degree k. Then:

(1) k is an eigenvalue of A;

(2) if G is connected then the multiplicity of k is one;

(3) for any eigenvalue λ of A, we have $|\lambda| \leq k$;

(4) there is an eigenvector $\underline{u}$ belonging to the eigenvalue k with all co-ordinates equal to one. In a connected graph, the eigenvectors belonging to the other eigenvalues are orthogonal to $\underline{u}$ and hence the sum of their co-ordinates is equal to zero.

Theorem 2.0.7

An irreducible non-negative matrix A cannot have two linearly independent non-negative eigenvectors.

Theorem 2.0.8

Suppose A is the adjacency matrix of a bipartite graph G. If A has eigenvalue λ of multiplicity $m(\lambda)$, then $-\lambda$ is also an eigenvalue of A, and $m(-\lambda) = m(\lambda)$.

Theorem 2.0.9

The adjacency matrix A of a connected graph G with n vertices and diameter d has at least d+1, and at most n distinct eigenvalues.

Theorem 2.0.10

If the spectrum of G contains eigenvalue λ with multiplicity p (where $p > 1$), then the spectrum of the complement $\bar{G}$ contains eigenvalue $\lambda - 1$ of multiplicity $\bar{p}$ satisfying the inequality: $\bar{p} \geq p - 1$.

[1] A complete bigraph $K_{m,n}$ is a bipartite graph with the set of vertices divided into two sets V_1 and V_2 having m and n points respectively and the graph contains every line joining elements of V_1 and elements of V_2 and no other lines.

2.1 The Relevance of Eigenvalues and Eigenvectors to Isomorphism

In order to maintain consistent notation and to avoid repetitive definition of symbols in this chapter and throughout the rest of this thesis we adopt the following conventions for symbols: G_1 and G_2 are two connected n-point cospectral graphs with adjacency matrices A_1 and A_2 respectively. Let $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ denote the complete set of eigenvalues (not necessarily distinct) of A_1 and A_2. Let $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ be the set of distinct eigenvalues of A_1 and A_2 where $k \leq n$ and $\alpha_1 < \alpha_2 < \alpha_3 < \ldots < \alpha_k$. Let m_i denote the multiplicity of α_i for $i=1,2,\ldots,k$ and $\sum_{j=1}^{k} m_j = n$. For $i=1,2,\ldots,k$ let V_i denote the eigenvector space associated with α_i for A_1 and similarily for $i=1,2,\ldots,k$ let V_i' be the eigenvector space associated with α_i for A_2.

A. Mowshowitz [1974] has suggested that in dealing with cospectral graphs, one might extract isomorphism-relevant information by examining the vector spaces of eigenvectors (hereafter called eigenvector spaces) associated with the eigenvalues. The following result gives a precise relationship between the vector spaces (corresponding to the same eigenvalue) of two isomorphic graphs.

Theorem 2.1.1 (Chao [1965])

Let G_1 and G_2 be isomorphic graphs and so $A_1 = P^T A_2 P$ where P is an nxn permutation matrix representing an isomorphism between G_1 and G_2. If $\underline{u}_i$ is an eigenvector of A_1 belonging to eigenvalue α_i then $P\underline{u}_i$ is an eigenvector of A_2 belonging to eigenvalue α_i.

Proof:

We have:

(i) $A_1\underline{u}_i = \alpha_i\underline{u}_i$

and (ii) $PA_1 = A_2P$ (since $A_1 = P^T A_2 P$).

Multiplying (i) by P, we have:

(iii) $PA_1\underline{u}_i = P\alpha_i\underline{u}_i$.

Now substituting (ii) in (iii) and since α_i is a constant, we have:

(iv) $A_2P\underline{u}_i = \alpha_i P\underline{u}_i$.

Hence, $P\underline{u}_i$ is an eigenvector of A_2 belonging to eigenvalue α_i.

QED.

Theorem 2.1.1 provides a stepping stone for solving graph isomorphism in that it indicates discriminating information about the eigenvector spaces of two non-isomorphic cospectral graphs. Before proceeding any further it is necessary to state the following definition: two eigenvector spaces V_i and V_i' both with eigenvectors of n components are said to be a permutation of each other iff there exists an nxn permutation matrix P such that $P\underline{u}_j \in V_i'$ for any eigenvector $\underline{u}_j \in V_i$ and $P^T\underline{u}_r' \in V_i$ for any eigenvector $\underline{u}_r' \in V_i'$. This will be denoted by $V_i^{(P)} = V_i'$ or $V_i = V_i'^{(P^T)}$. The following corollary of Theorem 2.1.1 relates this notion of eigenvector spaces being a permutation of one another to the graph isomorphism problem:

Corollary 2.1.2

If there does not exist an nxn permutation matrix P such that $V_i^{(P)} = V_i'$ for i=1,2,...,k then $G_1 \not\cong G_2$.

This corollary is of central importance since it provides a condition to determine the non-isomorphism (if this is the case) of two given cospectral graphs. The methods presented in this thesis examine the eigenvector spaces of the adjacency matrices of two given cospectral graphs G_1 and G_2 in an effort to discover whether or not the corresponding eigenvector spaces of the adjacency matrices from each graph are a permutation P of each other. If no such P exists then the graphs are not isomorphic by Corollary 2.1.2. However if a permutation matrix P is discovered with the property that $V_i^{(P)} = V_i'$ for i=1,2,...,k, then P represents an isomorphism between the two graphs. Gelbart [1976] proves this result:

Theorem 2.1.3 (Gelbart [1976])

If an nxn permutation matrix P exists such that $V_i^{(P)} = V_i'$ for i=1,2,.....,k then P represents an isomorphism from G_1 to G_2.

Proof:

Let D be the diagonal matrix with $d_{i,i} = \lambda_i$ where i=1,2,....,n and the eigenvalues λ_i are not necessarily distinct if $k < n$. Since A_1 and A_2 are simple matrices there exist orthogonal matrices R and Q such that:

$A_1 = RDR^T$ and $A_2 = QDQ^T$

where $R = [\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_n]$, the set $\{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_n\}$ are mutually orthogonal eigenvectors of A_1 and $\underline{x}_i$ belongs to λ_i for i=1,2,....,n. Similarily $Q = [\underline{y}_1, \underline{y}_2, \ldots, \underline{y}_n]$, the set $\{\underline{y}_1, \underline{y}_2, \ldots, \underline{y}_n\}$ are mutually orthogonal eigenvectors of A_2 and $\underline{y}_i$ belongs to λ_i for i=1,2,....,n.

Since $\underline{x}_i = P\underline{y}_i$ for i=1,2,...,n then $R = [P\underline{y}_1, P\underline{y}_2, \ldots, P\underline{y}_n] = PQ$.

Hence $A_1 = RDR^T = (PQ)D(PQ)^T = P(QDQ^T)P^T = PA_2P^T$.

Which is to say that P is an isomorphism between G_1 and G_2.

QED.

An algorithm to determine isomorphism based on this result must be able to detect whether or not there exists a permutation P between eigenvector spaces belonging to both simple and multiple eigenvalues. The case of the simple eigenvalue presents little obstacle; however, the multiple eigenvalues pose a greater challenge. In the next two sections we will discuss first of all how to deal with simple eigenvalues, and secondly the multiple eigenvalues.

2.2 Simple Eigenvalues

In order to determine whether or not G_1 and G_2 are isomorphic, the proposed method in this thesis examines their corresponding eigenvector spaces to determine if they are permutations of each other. If this is not the case then by Corollary 2.1.2 the graphs are not isomorphic. For the development of the algorithm in this section let us consider only the simple eigenvalues of A_1 and A_2. Suppose that α_j (where $j \in \{1,2,\ldots,k\}$) is one such simple eigenvalue. If $\underline{u} = (u_1, u_2, \ldots, u_n) \in V_j$ and similarily

$\underline{u}' = (u_1', u_2', \ldots, u_n') \in V_j'$, then the whole eigenvector spaces V_j and V_j' are generated by all real multiples of the eigenvectors $\underline{u}$ and $\underline{u}'$ respectively (since α_j is simple). So it follows that we need only examine the components of $\underline{u}$ and $\underline{u}'$ to see if they are multiples of one another as a set of numbers in order to determine if there is a permutation between them. Theorem 2.2.1 summarizes this notion:

Theorem 2.2.1

Let $\underline{x} = (x_1, x_2, \ldots, x_n) \in V_j$ and $\underline{y} = (y_1, y_2, \ldots, y_n) \in V_j'$ where α_j is a simple eigenvalue of A_1 and A_2. If there is no real number r and nxn permutation matrix P such that $\underline{x} = r.P\underline{y}$ then $G_1 \not\cong G_2$.

Proof:

Since α_j is simple, the eigenvector space V_j is generated by all the real multiples $r\underline{x}$ of the vector $\underline{x}$. Similarily the space V_j' is generated by all real multiples $t\underline{y}$ of the vector $\underline{y}$. Assume that $G_1 \cong G_2$. Then by Theorem 2.1.1 $V_j'^{(P)} = V_j$ and $\underline{x} = P\underline{z}$ where $\underline{z} \in V_j'$. Since α_j is simple then $\underline{z} = r\underline{y}$ for some real value r. Hence $\underline{x} = Pr\underline{y} = rP\underline{y}$. Therefore if no such real multiple exists then $G_1 \not\cong G_2$.

QED.

Lemma 2.2.2

If the vectors $\underline{x}$ and $\underline{y}$ of Theorem 2.2.1 are normalized and $\underline{x} \neq \pm P\underline{y}$ then $G_1 \not\cong G_2$.

Proof:

Let $\|\underline{x}\|$ denote the normalized vector of $\underline{x}$, then by definition $\|\underline{x}\| = (\frac{x_1}{b}, \frac{x_2}{b}, \ldots, \frac{x_n}{b})$ where $b = \sqrt{\sum_{i=1}^{n} x_i^2}$. If $\underline{z} = r\underline{y}$ where r is a real number then $\|\underline{z}\| = \pm \|\underline{y}\|$. Hence if the vectors are normalized in the proof of Theorem 2.2.1, then we need only consider whether the components of $\|\underline{x}\|$ are a +1 or a -1 multiple of some permutation of the components of $\|\underline{y}\|$.

QED.

If the graphs under consideration are connected and regular of degree s, then by Theorem 2.0.6 any eigenvector belonging to simple eigenvalue s has all components equal (in sign as well as magnitude). This being the

case, these eigenvectors will yield no helpful information to determine isomorphism in the context of Theorem 2.2.1 and Lemma 2.2.2. Consequently if the graph is regular of degree s then we may safely ignore the eigenvector space belonging to eigenvalue s.

For the rest of this thesis it is assumed that all vectors are normalized and consequently the symbol $\underline{x}$ will mean the normalized vector $\|\underline{x}\|$. Sometimes it is possible to determine that a given normalized vector is not a ± 1 multiple of another given normalized vector merely by noting the signs of the components. Lemma 2.2.3 indicates the pre-requisite condition:

Lemma 2.2.3

Let $\underline{x} = (x_1, x_2, \ldots, x_n)$ and $\underline{y} = (y_1, y_2, \ldots, y_n)$. Suppose that $\underline{x}$ has i positive components, i' zero components, and therefore n-i-i' negative components. Similarily assume that $\underline{y}$ has j positive components, j' zero components, and therefore n-j-j' negative components. First if $i' \neq j'$ then there does not exist a permutation matrix P such that $\underline{x} = \pm P\underline{y}$. Secondly, if i does not equal one of j or n-j-j' then there does not exist a permutation matrix P such that $\underline{x} = \pm P\underline{y}$.

Proof:

If $\underline{x} = \pm P\underline{y}$ then clearly $i' = j'$. Therefore if $i' \neq j'$ then $\underline{x} \neq \pm P\underline{y}$ for some permutation matrix P.

If $\underline{x} = P\underline{y}$ for some permutation matrix P then $i = j$. Otherwise if $\underline{x} = -P\underline{y}$ then $i = n-j-j'$. Hence if $i \neq j$ and $i \neq n-j-j'$ then $\underline{x} \neq \pm P\underline{y}$.

QED.

Lemma 2.2.3 combined with Lemma 2.2.2 provides a quick "pre-test" in the course of deciding the isomorphism of two cospectral graphs. As an illustration of this pre-test, consider the example of two cospectral 10-point trees shown in Figure 2.2.1.[2] The degree sequence of these trees is 3322221111 and their eigenvalues are ± 2.0886, ± 1.6810, ± 1.1491, ± 0.70108, 0, and 0. An eigenvector belonging to simple eigenvalue -2.0886

[2]The complete set of eigenvalues and eigenvectors for these graphs and the other graphs appearing in the figures of this chapter may be found in Appendix I.

for each tree is given in Figure 2.2.2.

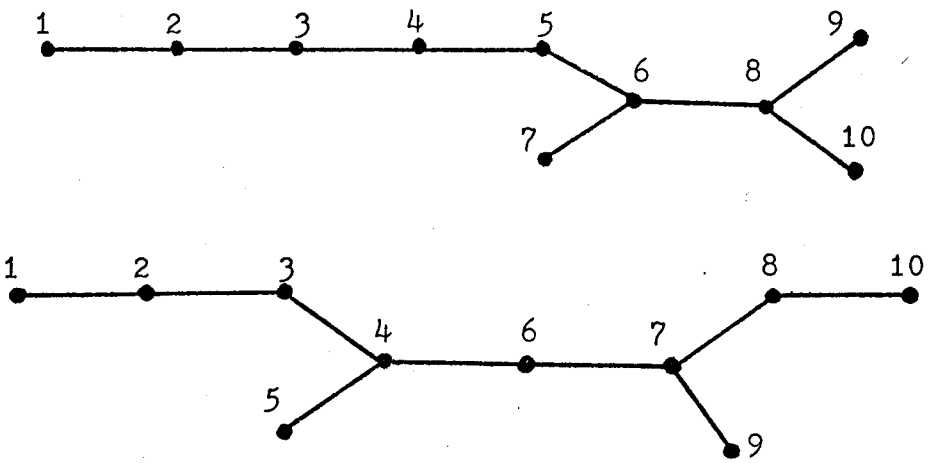


Figure 2.2.1 Two cospectral non-isomorphic 10-point trees.

tree(i)	tree(ii)
$\underline{x} \in V_1$	$\underline{y} \in V_1'$
-0.57158×10^{-1}	0.97990×10^{-1}
0.11938	-0.20466
-0.19219	0.32948
0.28202	-0.48349
-0.39685	0.23149
0.54685	0.44886
-0.26182	-0.45401
-0.48349	0.28202
0.23149	0.21737
0.23149	-0.13503

Figure 2.2.2 Eigenvectors belonging to eigenvalue -2.0886 for the trees in Figure 2.2.1.

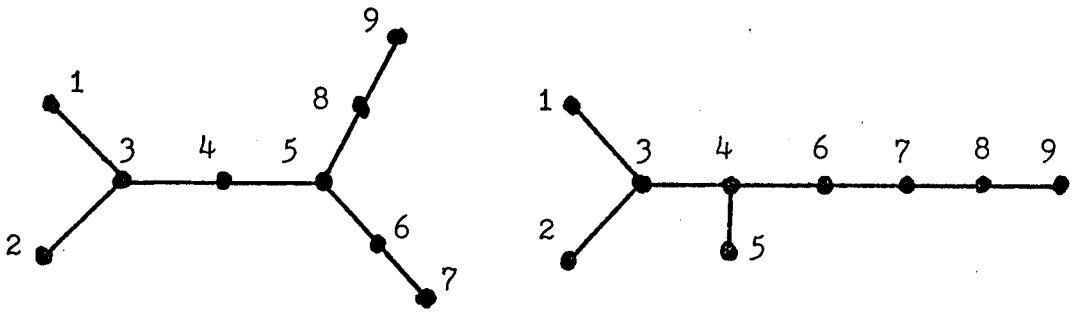
The first step in implementing the pre-test derived from Lemma 2.2.3 involves counting the number of positive components of each vector. Tree (i) has five positive components and therefore five negative components. Whereas tree (ii) has six positive components and therefore four negative components. Therefore by Lemma 2.2.3 $\underline{x} \neq \pm P\underline{y}$ and so we may conclude that the trees are not isomorphic.

Shortly the algorithm will be discussed more formally. First we present two further examples of non-isomorphic cospectral graphs of the same degree sequence that are recognized as being non-isomorphic via Theorem 2.2.1. The first example requires only the quick pre-test given in Lemma 2.2.3; whereas, the second example requires slightly more processing. Finally we show a pair of isomorphic trees that require the algorithm of Chapter 3 to search for the isomorphism.

The two trees shown in Figure 2.2.3 are the smallest non-isomorphic cospectral trees with the same degree sequence. The eigenvalues for these trees are ± 2.0840, ± 1.5718, ± 1.0000, ± 0.43173, and 0. An eigenvector belonging to simple eigenvalue -1.0 for each tree is given in Figure 2.2.3. The eigenvector for tree (i) has five zero components, and the eigenvector for tree (ii) has one zero component. Therefore by the quick pre-test we may conclude that tree (i) is not isomorphic to tree (ii).

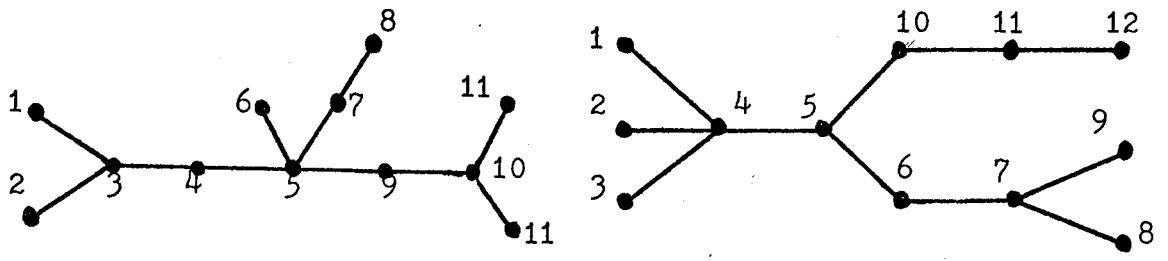
Two non-isomorphic trees (from Turner [1968]) are given in Figure 2.2.4 along with an eigenvector for each belonging to simple eigenvalue 1.7321. All the eigenvalues for these two graphs are: ± 2.2725, ± 1.7321, ± 1.4924, ± 0.78014, 0, 0, 0, and 0. The given eigenvectors both have four negative and four positive components and so either a +1 or a -1 multiple is possible. However $\{-0.28868, -0.50000, -0.28868, -0.28868\}$ is not equal to the set of positive components or the set of negative components from tree (ii). Therefore we may conclude that the

two graphs are not isomorphic.



tree (i)	tree (ii)
0.0	-0.35355
0.0	-0.35355
0.0	0.35355
0.0	0.35355
0.0	-0.35355
-0.50000	-0.35355
0.50000	0.0
0.50000	0.35355
-0.50000	-0.35355

Figure 2.2.3 Smallest non-isomorphic cospectral trees with the same degree sequence, and their eigenvectors belonging to simple eigenvalue -1.0.

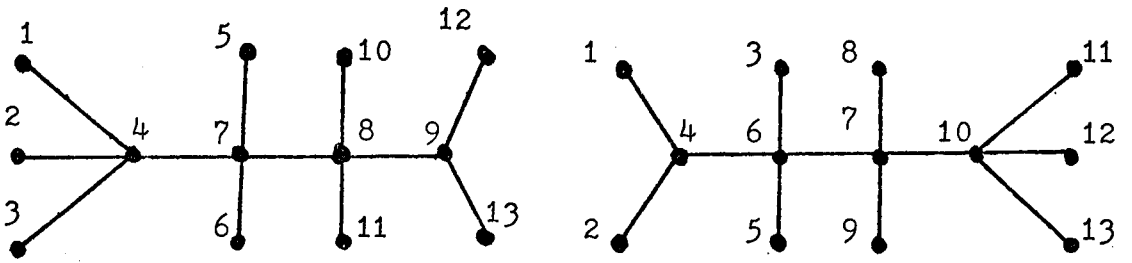


tree (i)	tree (ii)
0.28868	-0.20412
0.28868	-0.20412
0.50000	-0.20412
0.28868	-0.35355
0.0	0.0
0.0	0.35355
0.0	0.61237
0.0	0.35355
-0.28868	0.35355
-0.50000	0.0
-0.28868	0.0
-0.28868	0.0

Figure 2.2.4 Two non-isomorphic cospectral trees (from Turner [1968]) with eigenvectors belonging to simple eigenvalue 1.7321.

Two isomorphic trees are shown in Figure 2.2.5 together with an eigenvector for each tree belonging to simple eigenvalue -2.4915. Not surprisingly upon examination of all the eigenvectors belonging to simple eigenvalues the method of this section cannot reach a conclusion about the isomorphism of these graphs. Similarily the algorithm of section 2.3 will fail to reach an isomorphism decision upon examination of the eigenvectors belonging to the multiple eigenvalue. However we will see

that the algorithm of Chapter 3 will successfully determine an isomorphism of these trees using 0 backtracks.



tree (i)	tree (ii)
-0.16203	-0.11265
-0.16203	-0.11265
-0.16203	0.19023
0.40370	0.28065
0.20860	0.19023
0.20860	-0.47394
-0.51971	0.51971
0.47394	-0.20860
-0.28065	-0.20860
-0.19023	-0.40370
-0.19023	0.16203
0.11265	0.16203
0.11265	0.16203

Figure 2.2.5 Two isomorphic 13-point trees with eigenvectors belonging to simple eigenvalue -2.4915.

The present chapter focuses on determining the non-isomorphism of two given cospectral graphs while the actual search for an isomorphism is discussed in Chapter 3. For the moment Algorithm 2.2.1 is a summary of the procedure thus far with the simple eigenvalues.

Algorithm 2.2.1 The search for non-isomorphism through examination of the eigenvectors belonging to the simple eigenvalues.

It is assumed that G_1 and G_2 have the same degree sequence as well as being cospectral. Let $\underline{u}_i$ and $\underline{u}_i'$ be normalized eigenvectors belonging to α_i for A_1 and A_2 respectively.

Step 1: I:=0.

Step 2: I:=I+1.

Step 3: IF α_i is not a simple eigenvalue THEN step 12.

Step 4: IF the graphs are regular of degree $|\alpha_i|$ THEN step 12.

Step 5: Let J:=the number of positive components of $\underline{u}_i$.
Let J':=the number of zero components of $\underline{u}_i$.
Let Z:=the number of positive components of $\underline{u}_i'$.
Let Z':=the number of zero components of $\underline{u}_i'$.
IF J' $\neq$ Z' THEN TERMINATE since the graphs are not isomorphic.

Step 6: Sort the non-zero components of $\underline{u}_i$ into $\{x_1, x_2, \ldots, x_{n-j'}\}$ such that $x_1 \leq x_2 \leq \ldots \leq x_{n-j'}$.
Sort the non-zero components of $\underline{u}_i'$ into $\{y_1, y_2, \ldots, y_{n-j'}\}$ such that $y_1 \leq y_2 \leq \ldots \leq y_{n-j'}$.
IF J = Z THEN IF J = N-J-J' THEN step 10 ELSE step 9.

Step 7: IF J $\neq$ N-Z-Z' THEN TERMINATE since the graphs are not isomorphic.

STEP 8: A -1 multiple is the only possibility.
FOR W:=1 UNTIL N-J' DO IF $x_w \neq y_{n-j'-w+1}$ THEN TERMINATE since the graphs are not isomorphic.
Otherwise step 12.

Step 9: A +1 multiple is the only possibility.
FOR W:= 1 UNTIL N-J' DO IF $x_w \neq y_w$ THEN TERMINATE since the graphs are not isomorphic.
Otherwise step 12.

Step 10: Either a +1 or a -1 multiple is possible.
FOR W:=1 UNTIL N-J' DO IF $x_w \neq y_w$ THEN step 11.
Otherwise step 12.

Step 11: FOR W:=1 UNTIL N-J' DO IF $x_w \neq y_{n-j'-w+1}$ THEN TERMINATE since the graphs are not isomorphic.

Step 12: IF I $\neq$ K THEN step 2.
Otherwise the graphs are possibly isomorphic, so consider the eigenvector spaces belonging to the multiple eigenvalues using the method of section 2.3. If this does not resolve the problem then use the backtracking algorithm of Chapter 3 to search for an isomorphism.
TERMINATE.

The order of the computation for detecting non-isomorphism thus far, is roughly as follows:

(1) computing the eigenvalues and eigenvectors for both graphs requires $O(n^3)$ time (see Reinsch & Wilkinson [1971]).

(2) sorting the eigenvalues by a heapsort requires O(nlogn) time.

(3) counting the multiplicities of the eigenvalues requires O(n) comparisons.

(4) for each one of the w simple eigenvalues where $1 \leq w \leq k$:

  (i) counting the number of positive and zero components of both eigenvectors requires O(n) comparisons.

  (ii) sorting the components of $\underline{u}_i$ and $\underline{u}'_i$ by a heapsort requires O(nlogn) comparisons.

  (iii) the possibility of both a +1 and a -1 multiple is the worst case in comparing the sorted components. At worst this would require O(n) comparisons.

The preceding analysis predicts that the total computation is of order $n^3 + n\log n + n + w(n + n\log n)$. However the upper bound for w is k, which in turn has an upper bound of n, and so the computation for (4) above is of order $n^2 + n^2\log n$. Hence the overall computation is predicted to be of order $n^3 + n^2 + n^2\log n + n\log n + n$ at worst. In practice it was observed that if the graphs were not isomorphic that this was immediately detected upon examination of the first eigenvector. In this case the computation is of order $n^3 + n\log n + n$.

2.3 Multiple Eigenvalues

In the previous section it is observed that an examination of the eigenvectors belonging to the simple eigenvalues of a graph may determine the non-isomorphism of two cospectral graphs. If a "difference" cannot be detected upon examination of the eigenvectors belonging to the simple eigenvalues then it is necessary to process the eigenvector spaces belonging to the multiple eigenvalues. As an example consider the two regular bipartite graphs of degree 4 (due to Hoffman [1963]) shown in Figure 2.3.1. The eigenvalues of these graphs with their multiplicities shown in parentheses are: -4.0(1), -2.0(4), 0.0(6), 2.0(4), and 4.0(1). Since the graphs are regular of degree 4 the eigenvector belonging to eigenvalue 4.0 would yield no helpful information because the components are equal (see Theorem 2.0.6). Similarily for the eigenvector of -4.0 the components are equal in magnitude but half of them are negative. So in this particular example it is necessary to turn to the eigenvector spaces belonging to the multiple eigenvalues.

Comparing the vector spaces of multiple eigenvalues in a reasonable computation time presents quite a challenge for the following reason:

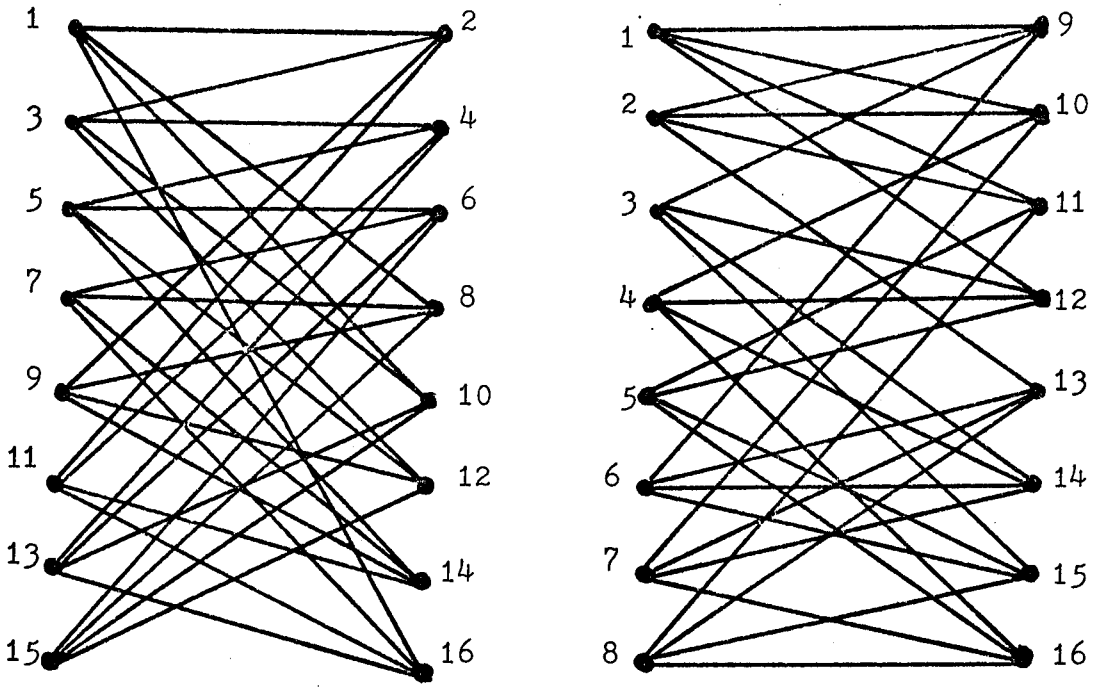


Figure 2.3.1 Two cospectral bipartite graphs of degree 4.

Since eigenvalue α_i has multiplicity $m_i > 1$, then by Theorem 2.0.3 V_i (the eigenvector space) is generated by any set of m_i linearly independent eigenvectors belonging to α_i for A_1. Similarily, V_i' is generated by any set of m_i linearly independent eigenvectors belonging to α_i for A_2. Therefore a given set of basis eigenvectors $\{\underline{e}_1, \underline{e}_2, \ldots, \underline{e}_{m_i}\}$ for V_i and a given set of basis eigenvectors for V_i' will not necessarily be a permutation of each other even if $V_i^{(P)} = V_i'$ for some permutation P. In computing a basis for V_i the only restriction imposed on the eigenvectors is that they must be any m_i normalized mutually orthogonal eigenvectors belonging to α_i for the given matrix. So the crux of the problem is to find a canonical form for comparing V_i and V_i' without having to generate the whole space as computing a basis for each eigenvector space will not indicate whether or not $V_i^{(P)} = V_i'$ for some permutation matrix P.

Let $\{\underline{e}_1, \underline{e}_2, \ldots, \underline{e}_n\}$ and $\{\underline{e}'_1, \underline{e}'_2, \ldots, \underline{e}'_n\}$ be sets of mutually orthogonal and normalized basis eigenvectors for A_1 and A_2 respectively where $\underline{e}_i$ and $\underline{e}'_i$ belong to λ_i for A_1 and A_2 respectively. The following result is essentially a re-statement of a theorem of matrix theory which states that a matrix is simple iff it is similar to a diagonal matrix. Without loss of generality the results are given in terms of A_1.

Theorem 2.3.1

A_1 is uniquely determined by any set of n mutually orthogonal eigenvectors belonging to $\lambda_1, \lambda_2, \ldots, \lambda_n$ respectively.

Proof:

If we define the diagonal matrix $D = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$ and the orthogonal matrix $E = \left[\underline{e}_1 \,\middle|\, \underline{e}_2 \,\middle|\, \ldots \,\middle|\, \underline{e}_n\right]$ then by the definition of eigenvalues and eigenvectors we have the matrix equation: $A_1E = ED$. However since E is non-singular this expression may be rewritten as: $A_1 = EDE^{-1}$.

QED.

For i=1,2,....,k let U_i denote the nxm_i matrix whose columns are normalized orthogonal eigenvectors of A_1 belonging to eigenvalue α_i. U'_i is similarily defined for A_2. Furthermore let $\{\underline{u}_{i1}, \underline{u}_{i2}, \ldots, \underline{u}_{im_i}\}$ be a set of normalized mutually orthogonal basis eigenvectors that constitute columns 1 through m_i of U_i. The set $\{\underline{u}'_{i1}, \underline{u}'_{i2}, \ldots, \underline{u}'_{im_i}\}$ is similarily defined for U'_i.

Note that $U_i^T.U_i = \begin{bmatrix} \underline{u}_{i1}^T \\ \underline{u}_{i2}^T \\ \vdots \\ \underline{u}_{im_i}^T \end{bmatrix} \left[\underline{u}_{i1} \,\middle|\, \underline{u}_{i2} \,\middle|\, \ldots \,\middle|\, \underline{u}_{im_i}\right] = I$

since the eigenvectors are normalized.

Theorem 2.3.2

Let $H_i = U_i U_i^T$ for $i=1,2,\ldots,k$. Then H_i is the unique symmetric matrix such that:

$$H_i\underline{x} = \begin{cases} \underline{x} & \text{for } \underline{x} \in V_i \\ \underline{0} & \text{for } \underline{x} \perp V_i. \end{cases}$$

Proof:

$$H_i\underline{x} = U_i U_i^T \underline{x} = \left[\underline{u}_{i1} \vdots \underline{u}_{i2} \vdots \cdots\cdots \vdots \underline{u}_{im_i}\right] \begin{bmatrix} \underline{u}_{i1}^T \\ \underline{u}_{i2}^T \\ \underline{u}_{i3}^T \\ \vdots \\ \underline{u}_{im_i}^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Consider first of all the product:

$$U_i^T\underline{x} = \begin{bmatrix} \underline{u}_{i1}^T \\ \underline{u}_{i2}^T \\ \vdots \\ \underline{u}_{im_i}^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \underline{u}_{i1}\cdot\underline{x} \\ \underline{u}_{i2}\cdot\underline{x} \\ \vdots \\ \underline{u}_{im_i}\cdot\underline{x} \end{bmatrix}$$

If $\underline{x} \perp V_i$ then $U_i^T.\underline{x} = \underline{0}$ and so the product $U_i U_i^T\underline{x} = \underline{0}$.

However if $\underline{x} \in V_i$ there exists a set of real numbers $\{\gamma_1, \gamma_2, \ldots, \gamma_{m_i}\}$ such that $\underline{x} = \gamma_1\underline{u}_{i1} + \gamma_2\underline{u}_{i2} + \cdots\cdots + \gamma_{m_i}\underline{u}_{im_i}$. In this case the product $U_i^T\underline{x} = \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \vdots \\ \gamma_{m_i} \end{bmatrix}$ and therefore

$$U_i U_i^T\underline{x} = \left[\underline{u}_{i1} \vdots \underline{u}_{i2} \vdots \cdots\cdots \vdots \underline{u}_{im_i}\right] \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{m_i} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \underline{x}.$$

It is easily seen that H_i is symmetric for i=1,2,...,k. Because of the above development each H_i matrix has eigenvalue 1 for eigenvector $\underline{x} \in V_i$ and eigenvalue 0 for $\underline{x} \perp V_i$. Since the vectors $\underline{u}_{i1}, \underline{u}_{i2}, \ldots\ldots\ldots, \underline{u}_{im_i}$ are mutually orthogonal and belong to eigenvalue 1 for the matrix H_i, then the real symmetric matrix H_i is unique by Theorem 2.3.1.

QED.

Note that if H_i corresponds to the maximum eigenvalue of a regular graph then it is an nxn symmetric matrix with all identical entries (since the eigenvalue is simple and an eigenvector belonging to α_i has all identical components).

Hereafter by the matrix decomposition of A_1 we shall mean the expression $A_1 = \sum_{i=1}^{k} \alpha_i H_i$ where $\alpha_1 < \alpha_2 < \ldots\ldots < \alpha_k$ are the distinct eigenvalues of A_1 and $H_i = U_i U_i^T$ as in Theorem 2.3.2.

Theorem 2.3.3

The H_i's (for i=1,2,...,k) are idempotent.

Proof:

$$H_i^2 = \left[\underline{u}_{i1} \vdots \underline{u}_{i2} \vdots \ldots\ldots \vdots \underline{u}_{im_i}\right] \begin{bmatrix} \underline{u}_{i1}^T \\ \underline{u}_{i2}^T \\ \vdots \\ \underline{u}_{im_i}^T \end{bmatrix} \left[\underline{u}_{i1} \vdots \underline{u}_{i2} \vdots \ldots\ldots \vdots \underline{u}_{im_i}\right] \begin{bmatrix} \underline{u}_{i1}^T \\ \underline{u}_{i2}^T \\ \vdots \\ \underline{u}_{im_i}^T \end{bmatrix}$$

$$= \left[\underline{u}_{i1} \vdots \underline{u}_{i2} \vdots \ldots\ldots \vdots \underline{u}_{im_i}\right] \left[I\right] \begin{bmatrix} \underline{u}_{i1}^T \\ \underline{u}_{i2}^T \\ \vdots \\ \underline{u}_{im_i}^T \end{bmatrix} = H_i.$$

QED.

For $i=1,2,\ldots,k$ let $H_i' = U_i'U_i'^T$ denote the matrices in the matrix decomposition of A_2.

Theorem 2.3.4

$G_1 \cong G_2$ iff there exists a permutation matrix P such that $H_i = P^T H_i' P$ for all $i=1,2,\ldots,k$.

Proof:

(i) Assume $G_1 \cong G_2$.
Then by Theorem 2.1.1 $V_i^{(P)} = V_i'$ for $i=1,2,\ldots,k$ where P is a permutation matrix representing an isomorphism between G_1 and G_2. Therefore there exists a set of orthogonal basis eigenvectors for V_i which form the columns of U_i and a set of orthogonal basis eigenvectors for V_i' which form the columns of U_i' such that $U_i = P^T U_i'$.
Thus we have:

$$\begin{aligned} H_i &= U_i U_i^T \\ &= P^T U_i' (P^T U_i')^T \\ &= P^T U_i' U_i'^T P \\ &= P^T H_i' P \end{aligned}$$

(since H_i' is independent of the choice of basis for V_i').

(ii) Suppose $H_i = P^T H_i' P$ for $i=1,2,\ldots,k$.
The matrix decompositions of A_1 and A_2 are:

$$A_1 = \alpha_1 H_1 + \alpha_2 H_2 + \ldots\ldots\ldots + \alpha_k H_k \quad \text{and}$$

$$A_2 = \alpha_1 H_1' + \alpha_2 H_2' + \ldots\ldots\ldots + \alpha_k H_k'.$$

If we multiply the matrix decomposition of A_2 on the left by P^T and on the right by P, the expression becomes:

$$\begin{aligned} P^T A_2 P &= \alpha_1 P^T H_1' P + \alpha_2 P^T H_2' P + \ldots\ldots\ldots + \alpha_k P^T H_k' P \\ &= A_1 \quad (\text{since } H_i = P^T H_i' P). \end{aligned}$$

Hence $G_1 \cong G_2$ by an isomorphism represented by P.

QED.

Let A_1 and A_2 be two simple symmetric matrices with the same eigenvalues of the same multiplicity. Let $A_1 = \sum_{i=1}^{k} \alpha_i H_i$ and $A_2 = \sum_{i=1}^{k} \alpha_i H_i'$ be their respective matrix decompositions. Let us regard the columns of

H_i (or alternatively the rows since H_i is symmetric) as n sets of numbers. If for i=1,2,...,k the matrix H_i' has the same n sets of numbers as matrix H_i for its columns (or rows) then we will say that A_1 and A_2 have <u>similar matrix decompositions</u>.

<u>Lemma 2.3.5</u>

If A_1 and A_2 do not have similar matrix decompositions then $G_1 \not\cong G_2$.

Proof:

Since A_1 and A_2 do not have similar matrix decompositions there exists at least one $j \in \{1,2,\ldots,k\}$ such that the n sets of numbers in the columns of H_j are not the same as the n sets of numbers in the columns of H_j'. Therefore $H_j \neq P^T H_j' P$ for any nxn permutation matrix P. Hence by Theorem 2.3.4 $G_1 \not\cong G_2$.

QED.

By an <u>automorphism</u> of a matrix A_1 we mean a permutation matrix P with the property that $A_1 = P^T A_1 P$. It is trivial to verify that the set of all such automorphisms form a group which we shall call the <u>automorphism group</u> of the matrix A_1.

<u>Theorem 2.3.6</u>

The automorphism group of A_1, $\Gamma_{A_1} = \bigcap_{i=1}^{k} \Gamma_{H_i}$.

Proof:

The matrix decomposition of A_1 is:

$A_1 = \alpha_1 H_1 + \alpha_2 H_2 + \ldots\ldots + \alpha_k H_k$.

(i) Assume $P_\sigma \in \Gamma_{A_1}$. This implies that $P_\sigma^T A_1 P_\sigma = A_1$. From the matrix decomposition of A_1 we have:

$$P_\sigma^T A_1 P_\sigma = \alpha_1 P_\sigma^T H_1 P_\sigma + \alpha_2 P_\sigma^T H_2 P_\sigma + \ldots\ldots + \alpha_k P_\sigma^T H_k P_\sigma$$

$$= A_1.$$

However by Theorem 2.3.2 the H_i matrices are unique. Therefore $P_\sigma^T H_i P_\sigma = H_i$ for i=1,2,...,k, which is to say that $P_\sigma \in \bigcap_{i=1}^{k} \Gamma_{H_i}$.

(ii) Let $P_\sigma \in \bigcap_{i=1}^{k} \Gamma_{H_i}$. This means that $H_i = P_\sigma^T H_i P_\sigma$ for i=1,2,...,k.

If we multiply the expression for the matrix decomposition of A_1 on

the left by P_σ^T and on the right by P_σ we get:

$$P_\sigma^T A_1 P_\sigma = \alpha_1 P_\sigma^T H_1 P_\sigma + \alpha_2 P_\sigma^T H_2 P_\sigma + \ldots\ldots + \alpha_k P_\sigma^T H_k P_\sigma$$

$$= \alpha_1 H_1 + \alpha_2 H_2 + \ldots\ldots + \alpha_k H_k \quad (\text{since } H_i = P_\sigma^T H_i P_\sigma)$$

$$= A_1.$$

Hence $P_\sigma \in \Gamma_{A_1}$.

QED.

Lemma 2.3.7

If a graph G_1 is transitive, then every column (or alternatively every row) of H_i has the same set of real numbers for i=1,2,....,k.

Proof:

Any automorphism $\sigma \in \Gamma_{A_1}$ effectively maps the columns of A_1 onto each other (with the appropriate permutation of the rows) so that A_1 remains unchanged after the permutation. Hence if G_1 is transitive, this means that there exists automorphisms in Γ_{A_1} that map any column t of A_1 onto any column j of A_1 (with the appropriate row interchanges). Such an automorphism would be an automorphism of H_i for i=1,2,...,k (by Theorem 2.3.6) that maps column t of H_i onto column j of H_i (with the appropriate row interchanges). Hence every column of H_i would have the same set of real numbers for i=1,2,...,k.

QED.

Lemma 2.3.8

Any isomorphism from G_1 to G_2 would map columns of H_i onto only those columns of H_i' with the same diagonal element. Furthermore, if a graph G_1 is transitive then any H_i matrix from the matrix decomposition of A_1 has all diagonal elements identical.

Proof:

Suppose $A_2 = P^T A_1 P$ where P represents an isomorphism between G_1 and G_2. Then by Theorem 2.3.4:

$H_i' = P^T H_i P$ for i=1,2,...,k.

Furthermore, one of the effects of the multiplication $P^T H_i P$ is to rearrange the diagonal elements of H_i in constructing the diagonal elements of H_i'. More explicitly, if $P: x \rightarrow y$ then the x^{th} diagonal element of H_i becomes the y^{th} diagonal element of H_i' under the above multiplication. Hence in looking for an isomorphism between G_1 and G_2 we need

only consider ones that would "map" any diagonal element from H_i onto a diagonal element with the same value in H_i'.

From this it follows directly that if G_1 is transitive then all the H_i matrices in the matrix decomposition of A_1 have equal entries down the main diagonal.

QED.

How is all of this information used to determine non-isomorphism or isomorphism (as the case may be) of two given cospectral graphs? First of all, Theorems 2.3.2 and 2.3.4 indicate an invariant H_i for V_i, and a necessary and sufficient condition for isomorphism given any set of normalized orthogonal eigenvectors that form a basis for V_i. Briefly, an algorithm to determine isomorphism based on the results of this section would compute the H_i and H_i' matrices for A_1 and A_2 and compare their columns as sets of numbers in order to determine if space V_i is a permutation of space V_i'. If the sets of numbers are different then by Lemma 2.3.5 the graphs are not isomorphic. By Lemma 2.3.8, a quick scan comparing only the diagonal elements may be sufficient to detect non-isomorphism in some cases. If a difference between the two cospectral graphs is not detected using these techniques then the algorithm of Chapter 3 is employed either to disprove isomorphism or to find an isomorphism.

An examination of the H_i and H_i' matrices for the two bipartite graphs in Figure 2.3.1 would reveal that the first graph is possibly transitive and the second graph is definitely not transitive by Lemma 2.3.7 since the H_i' matrices do not have the same sets of numbers in every column. Furthermore the graphs are not isomorphic by Lemma 2.3.5 since the columns of the matrices H_i and H_i' that belong to eigenvalue -2.0 do not have the same sets of real numbers. The complete set of H_i and H_i' matrices for these graphs may be found in Appendix I between pages 157 and 167.

Another example of two non-isomorphic cospectral graphs that are discerned as non-isomorphic by this technique is given in Figure 2.3.2. The H_i matrices for these graphs are given in Appendix I. Note that for multiple eigenvalue 2.0 the H_i matrix has $\pm$.2500 in every column whereas some columns of H_i' have only +.2500. Hence by Lemma 2.3.5 the graphs are not isomorphic.

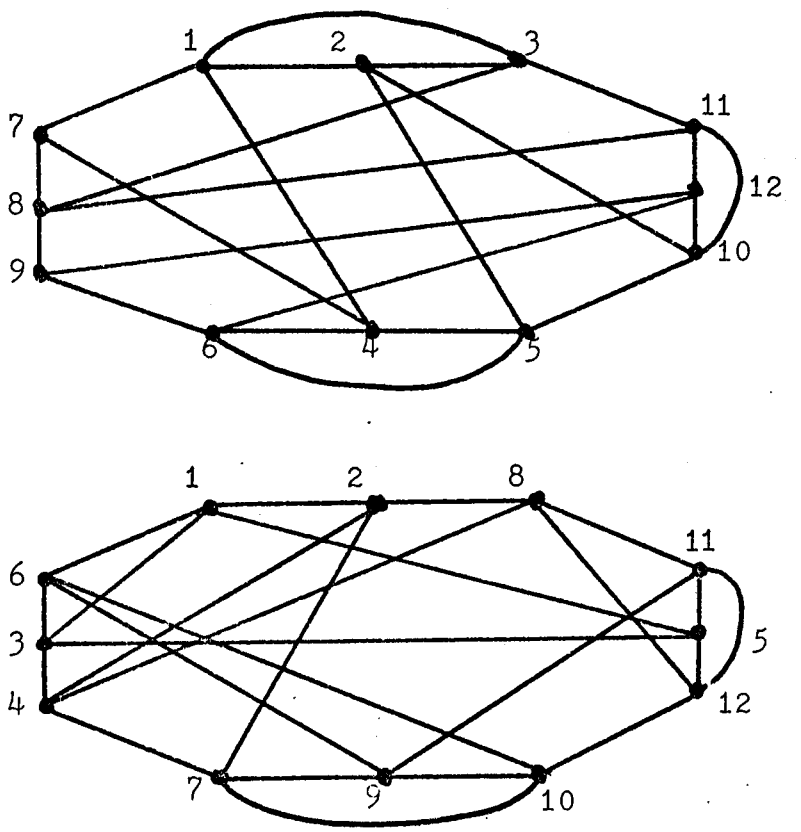


Figure 2.3.2 Two 12-point non-isomorphic cospectral graphs with eigenvalues 4.0(1), 0.0(3), 2.0(3), and -2.0(5).

A more formal description of the procedure thus far is given by Algorithm 2.3.1:

Algorithm 2.3.1 Processing the eigenvectors belonging to multiple eigenvalues in order to detect non-isomorphism.

Step 1: I:=0.

Step 2: I:=I+1.

Step 3: IF α_i is not a simple eigenvalue THEN step 4.
IF the graphs are not isomorphic by Algorithm 2.2.1 THEN TERMINATE ELSE step 6.

Step 4: Compute $H_i := U_i U_i^T$ and $H_i' := U_i' U_i'^T$.
IF the set of diagonal elements in H_i is not the same as the set of diagonal elements in H_i' THEN TERMINATE (since the graphs are not isomorphic by Lemma 2.3.8).

Step 5: Heapsort the numbers in each of the n columns for H_i and H_i'.
IF the n sets of numbers from H_i are not the same as the n sets of numbers from H_i' THEN TERMINATE (since the graphs are not isomorphic by Lemma 2.3.5).

Step 6: IF I $\neq$ K THEN step 2.

Step 7: The graphs are possibly isomorphic.
Use the backtracking algorithm of Chapter 3 to resolve this.
TERMINATE.

The order of the computation for this part of the method is roughly as follows:

(1) computing the eigenvalues and eigenvectors for both graphs requires $O(n^3)$ time (see Reinsch and Wilkinson [1971]).

(2) sorting the eigenvalues by a heapsort requires O(nlogn) comparisons.

(3) for each one of the t multiple eigenvalues of multiplicity $m_1,\ldots,m_t$:

(i) computing H_i and H_i' requires $\frac{n^2}{2}m_i$ multiplications.

(ii) comparing the diagonal elements requires order nlogn sorting time plus n comparisons.

(iii) sorting the sets of numbers in the columns of H_i and H_i' requires order n^2logn comparisons and ordering the sorted columns requires another order n^2logn comparisons.

(iv) comparing the sorted columns of H_i and H_i' requires at most n^2 comparisons.

Hence the total order of the computation is

$$n^3 + \text{nlogn} + \sum_{i=1}^{t} (\frac{n^2}{2}m_i + n^2\text{logn} + n^2 + \text{nlogn} + n)$$

However since t (the number of multiple eigenvalues) is bounded by n/2 and $\sum_{i=1}^{t} m_i$ is bounded by n, the order of the whole computation is:

$$n^3 + \text{nlogn} + n^3 + n^3\text{logn} + n^3 + n^2\text{logn} + n^2$$
$$\approx n^3\text{logn} + n^3 + n^2\text{logn} + n^2 + \text{nlogn}.$$

It is clear that the algorithm is capable of detecting the non-isomorphism of graphs with eigenvalues of high multiplicity in $O(n^3\text{logn})$ time. In some non-isomorphic cases however, further processing is necessary as will be discussed in Chapter 3.

CHAPTER 3

An Extension of the Algorithm to Handle "Difficult" Examples

3.0 Limitations of the Algebraic Filter

As was observed in Chapter 2, the matrix decomposition for the adjacency matrix of a graph provides us with an efficient method of determining the non-isomorphism of two cospectral graphs in $n^3 \log n$ time for some cases. It would seem very tempting to conjecture that if the adjacency matrices of two connected cospectral graphs G_1 and G_2 have similar matrix decompositions then $G_1 \cong G_2$. However, one does not have to search too far to discover that this is not the case. The twenty-five point bipartite cospectral graphs shown in Figure 3.0.1 (due to Mathon (reported in Corneil [1974])) form a counter-example to Corneil's conjecture. The adjacency matrices of these graphs have similar matrix decompositions and yet they are not isomorphic. The eigenvalues with their multiplicities of these 25-point graphs are: -4.899(1), -2.000(9), 0.0(5), 2.000(9), 4.899(1). Similarily, there exists a set of ten strongly regular 26-point graphs and a set of eight strongly regular 25-point graphs, each of which have adjacency matrices with similar matrix decompositions, and yet they are not isomorphic.

For these "more difficult" examples it is necessary to do some further processing in order to discern the non-isomorphic ones. Accordingly, a development which permits us to distinguish the non-isomorphism of these "difficult cases" and which provides us with a means of finding an isomorphism (if there is one), is presented in the rest of this chapter.

The results of Chapter 2 were explained in terms of the adjacency matrix of a graph; however, they are in fact relevant to any irreducible

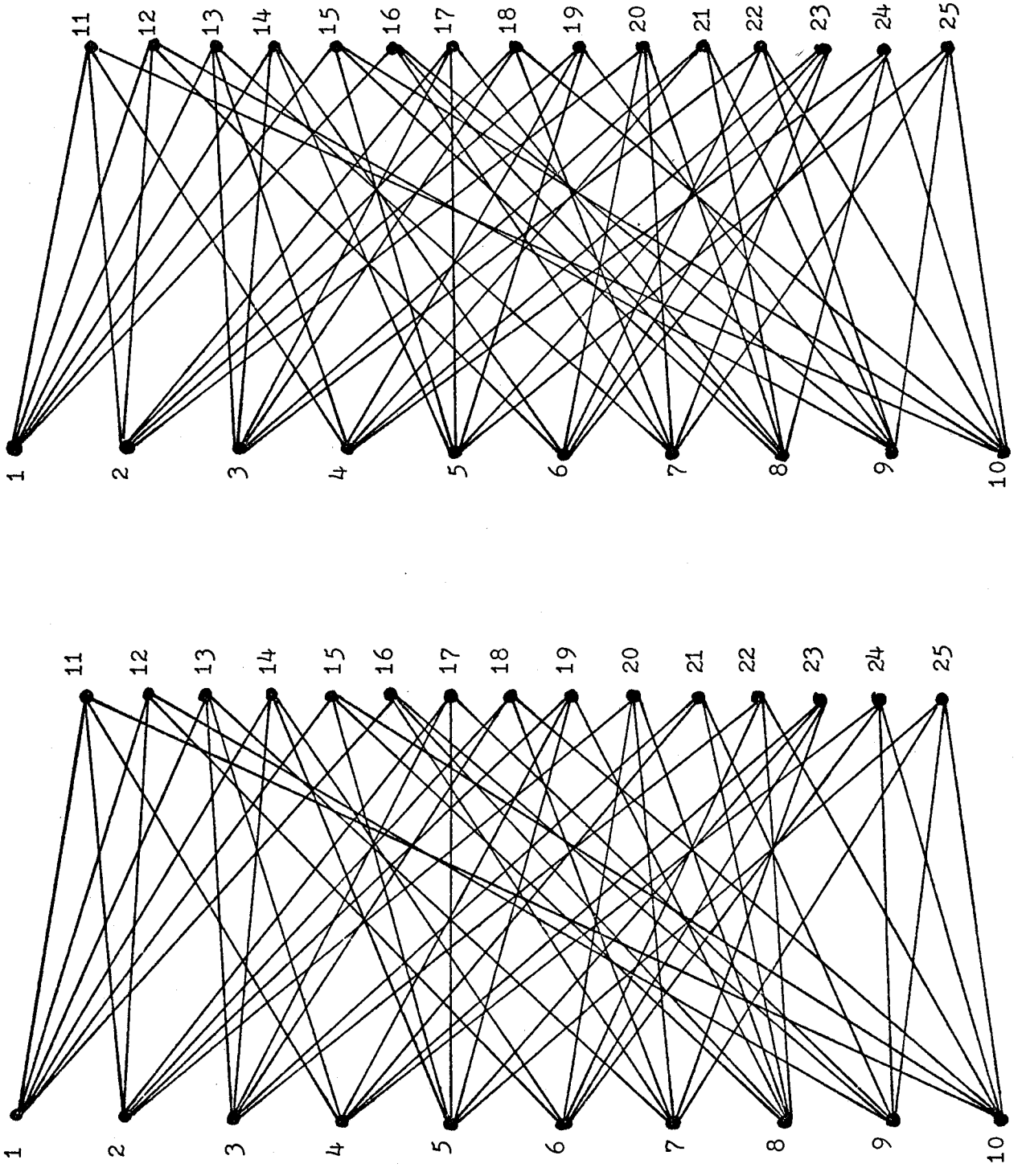


Figure 3.0.1 The 25-point counter-example to Corneil's conjecture.

real symmetric matrix and the proof depends only on those conditions and the fact that the eigenvectors are normalized. Whenever we have stated $G_1 \cong G_2$, this may be replaced by $A_1 = P^T A_2 P$ where A_1 and A_2 are irreducible real symmetric matrices and P is a permutation matrix. Accordingly, the algorithms of Chapter 2 may be applied to determine the non-isomorphism of not only graphs but also any pair of irreducible real symmetric matrices. It is in these more general terms that the theorems of this chapter are stated.

At this point it is necessary to state the following definitions: Two matrices are cospectral iff they have the same eigenvalues with the same multiplicities. By an isomorphism of two matrices A_1 and A_2 we mean a permutation σ represented by a permutation matrix P with the property that $A_1 = P^T A_2 P$. If such a P exists then we will say that the matrices A_1 and A_2 are isomorphic. By an automorphism of a matrix A_1 we mean a permutation σ represented by a permutation matrix P with the property that $A_1 = P^T A_1 P$. The set of all such automorphisms form the automorphism group of the matrix A_1.

3.1 Formation of the Type Matrix $T_\sigma(A_1^*)$

In this section a method is presented for "collapsing" the isomorphism relevant information from the matrix decomposition of A_1 into a more compact form which will be processed either to find the automorphism group generators of A_1 or to determine the existence or non-existence of an isomorphism between A_1 and another matrix. This information will be embodied in an nxn real symmetric "type" matrix which is constructed uniquely from the matrix A_1 and has the same automorphism group.

Recall from Chapter 2 that the matrix decomposition of the matrix A_1 may be written as: $A_1 = \alpha_1 H_1 + \alpha_2 H_2 + \ldots\ldots + \alpha_k H_k$ where the α_i's

are the distinct eigenvalues of A_1 ($\alpha_1 < \alpha_2 < \ldots\ldots < \alpha_k$) and the H_i's are matrices formed from $U_i U_i^T$ as in Theorem 2.3.2. If $a_{1_{i,j}}$ denotes the i,j^{th} element of matrix A_1, and $h_{k_{i,j}}$ denotes the i,j^{th} element of matrix H_k, then from the matrix decomposition of A_1 we may write:

$a_{1_{i,j}} = \alpha_1 h_{1_{i,j}} + \alpha_2 h_{2_{i,j}} + \ldots\ldots\ldots + \alpha_k h_{k_{i,j}}$ for $\forall\, i,j \in \{1,2,\ldots,n\}$.

If $a_{1_{i,j}} = \alpha_1 h_{1_{i,j}} + \alpha_2 h_{2_{i,j}} + \ldots\ldots\ldots + \alpha_k h_{k_{i,j}}$ (for $i,j \in \{1,2,..,n\}$)

and $a_{1_{r,s}} = \alpha_1 h_{1_{r,s}} + \alpha_2 h_{2_{r,s}} + \ldots\ldots\ldots + \alpha_k h_{k_{r,s}}$ (for $r,s \in \{1,2,..,n\}$)

where $h_{t_{i,j}} = h_{t_{r,s}}$ for $t=1,2,\ldots,k$ then the elements $a_{1_{i,j}}$ and $a_{1_{r,s}}$ of the matrix A_1 are said to be of the same <u>type</u>. Observe that all the elements of the same type from A_1 can be mapped uniquely onto a k-tuple of real numbers. Element $a_{1_{i,j}}$ is mapped onto the k-tuple $(h_{1_{i,j}}, h_{2_{i,j}}, \ldots\ldots, h_{k_{i,j}})$. Hereafter this k-tuple shall be called the <u>k-vector type</u> of element $a_{1_{i,j}}$. By definition, every element of the same type in A_1 is mapped onto the same k-tuple and therefore has the same k-vector type. Note that since the H_i's are symmetric, the element $a_{1_{i,j}}$ is automatically of the same type as $a_{1_{j,i}}$.

Before proceeding with further development we present two more definitions. Let A_1^* denote the nxn symmetric matrix of k-tuples where the i,j^{th} entry, $a^*_{1_{i,j}}$ is the k-vector type of $a_{1_{i,j}}$ from A_1, then A_1^* is called the <u>k-vector type matrix</u> of A_1 (see Algorithm 3.0). From A_1^*, the nxn symmetric <u>type matrix</u> $T_\delta^\gamma(A_1^*)$ is formed as follows:

Suppose that over the whole of the matrix A_1^* there exist z different k-tuples, then accordingly a bijection δ from the z different k-tuples onto a set γ of z different numbers may be defined. Note that any set of z different real and/or integer numbers will suffice. It is convenient to choose $\gamma = \{1,2,3,\ldots,z\}$ the set of consecutive integers from 1 to z, and so $T_\delta(A_1^*)$ will denote our type matrix. Generally the order in which

the integers are assigned to the elements of A_1^* does not matter as long as a given k-tuple is always mapped onto the same integer; however, in order to compare the type matrices of two given cospectral matrices, the same mapping δ must be used to define the respective type matrices. Algorithm 3.0 summarizes the formation of the k-vector type matrix and later Algorithm 3.1 will state how to form the type matrix $T_\delta(A_1^*)$ for any given A_1^*. Note that whenever we write $T_\delta(A_1^*)$ and $T_\delta(A_2^*)$ it is implied that they were constructed using the same mapping δ from the set of k-tuples uniquely to the set of integers from 1 to z.

Algorithm 3.0 Formation of the k-vector type matrix A_1^*

The algorithm assumes an irreducible real symmetric nxn matrix A_1 and matrix decomposition $A_1 = \alpha_1 H_1 + \alpha_2 H_2 + \ldots\ldots + \alpha_k H_k$.

Step 1: FOR i:=1 UNTIL n DO
FOR j:=1 UNTIL i DO
$a^*_{1\,i,j} := a^*_{1\,j,i} := (h_{1\,i,j}, h_{2\,i,j}, \ldots\ldots, h_{k\,i,j})$.

Step 2: TERMINATE.

The next step in the method is to define a bijection δ which maps the set of z different k-tuples in A_1^* onto the consecutive integers 1 through z. Conceptually the algorithm will examine the k-tuples of A_1^* one at a time, and whenever a new k-tuple $a^*_{1\,d,e}$ (for $d,e \in \{1,2,\ldots,n\}$) is encountered, it will be mapped onto the next "free" integer i_j in the set $\{1,2,\ldots,z\}$. After this the next "free" integer becomes i_j+1. Whenever a k-tuple equal to $a^*_{1\,d,e}$ is encountered again, δ will map it onto the same integer i_j as $a^*_{1\,d,e}$. Recall that A_1^* is symmetric, so it is only necessary to process half the matrix (our algorithm processes the lower triangle).

It is hoped that the following simple example will serve to illustrate the above concepts: Let $A_1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$ be our initial matrix. It

has four distinct eigenvalues: -1.5616, -1.0, 0.0 and 2.5616. So in this case k=4 and the matrix decomposition of A_1 is:

$A_1 = -1.5616*H_1 + (-1.0)*H_2 + 0.0*H_3 + 2.5616*H_4$ where

$$H_1 = \begin{bmatrix} 0.18836 & -0.24254 & 0.18936 & -0.24254 \\ -0.24254 & 0.31064 & -0.24254 & 0.31064 \\ 0.18936 & -0.24254 & 0.18936 & -0.24254 \\ -0.24254 & 0.31064 & -0.24254 & 0.31064 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 0.50000 & 0.0 & -0.50000 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ -0.50000 & 0.0 & 0.50000 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$H_3 = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.50000 & 0.0 & -0.50000 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.50000 & 0.0 & 0.50000 \end{bmatrix}$$

$$H_4 = \begin{bmatrix} 0.31064 & 0.24254 & 0.31064 & 0.24254 \\ 0.24254 & 0.18936 & 0.24254 & 0.18936 \\ 0.31064 & 0.24254 & 0.31064 & 0.24254 \\ 0.24254 & 0.18936 & 0.24254 & 0.18936 \end{bmatrix}$$

The k-vector type matrix A_1^* for the above matrix decomposition is shown in Figure 3.1.1. The type matrix $T_\delta(A_1^*)$ for the k-vector type matrix shown in Figure 3.1.1 is:

$$T_\delta(A_1^*) = \begin{bmatrix} 1 & 2 & 4 & 2 \\ 2 & 3 & 2 & 5 \\ 4 & 2 & 1 & 2 \\ 2 & 5 & 2 & 3 \end{bmatrix}$$

Although the method is described as above, it is possible to implement an equivalent process iteratively and directly from the matrix decomposition of A_1 in a more efficient manner. Instead of comparing whole k-tuples at a time, it is possible to build a type matrix itera-

.18936,0.50000,0.0,0.31064)	(-0.24254,0.0,0.0,0.24254)	(0.18936,-0.50000,0.0,0.31064)	(-0.24254,0.0,0.0,0.24254)
0.24254,0.0,0.0,0.24254)	(0.31064,0.0,0.50000,0.18936)	(-0.24254,0.0,0.0,0.24254)	(0.31064,0.0,-0.50000,0.18936)
.18936,-0.50000,0.0,0.31064)	(-0.24254,0.0,0.0,0.24254)	(0.18936,0.50000,0.0,0.31064)	(-0.24254,0.0,0.0,0.24254)
0.24254,0.0,0.0,0.24254)	(0.31064,0.0,-0.50000,0.18936)	(-0.24254,0.0,0.0,0.24254)	(0.31064,0.0,0.50000,0.18936)

Figure 3.1.1 The A_1^* matrix for the example in this section.

tively by comparing the i^{th} components of the k-tuples at a time and refining the type matrix. There is an initial type matrix T_1 which is formed uniquely from A_1 by putting a 1 in every place in T_1 where a 0 occurs in A_1, and a 2 every place in T_1 where a 1 occurs in A_1. So T_1 is a different form of the adjacency matrix in which there are only two initial types - type 1 corresponding to a 0 in A_1 and type 2 corresponding to a 1 in A_1. For the example given above $T_1 = \begin{bmatrix} 1 & 2 & 2 & 2 \\ 2 & 1 & 2 & 1 \\ 2 & 2 & 1 & 2 \\ 2 & 1 & 2 & 1 \end{bmatrix}$.

The initial type matrix is iteratively refined based on the information contained in the H_i matrices for i=1,2,...,k. Initially H_1 is used to refine T_1 which then becomes T_2, and more generally H_j is used to refine matrix T_j to matrix T_{j+1} (for j=1,2,....,k). When this process is finished, $T_\zeta(A_1^*) = T_{k+1}$ is the final type matrix of the initial matrix A_1. Note that since T_1 and H_i (for i=1,2,...,k) are symmetric matrices it is only necessary to process the lower triangles of the matrices. On each iteration the refinement of the matrix that occurs is roughly as follows: suppose we are beginning the j^{th} iteration of the process (ie. matrix H_j is scanned together with matrix T_j to form T_{j+1}). First of all, the integer-real pair $(t_{j_{1,1}}, h_{j_{1,1}})$ is mapped onto the integer 1 in T_{j+1} (that is to say $t^{j+1}_{1,1} = 1$). The occurrence of the pair $(t_{j_{1,1}}, h_{j_{1,1}})$ is remembered together with the fact that the corresponding entry in T_{j+1} was set to the integer 1. In this case out "next free" integer for this iteration is the number 2. The pair $(t_{j_{2,1}}, h_{j_{2,1}})$ is considered next. If $(t_{j_{2,1}}, h_{j_{2,1}}) = (t_{j_{1,1}}, h_{j_{1,1}})$ then $t^{j+1}_{2,1} = 1$; otherwise, $t^{j+1}_{2,1} = 2$, 3 becomes the "next free" integer, and this new mapping is remembered. The process continues in this fashion. For each d,e pair $d,e \in \{1,2,\ldots,n\}$ if $(t_{j_{d,e}}, h_{j_{d,e}})$ has not occurred before in this iteration then $t^{j+1}_{d,e}$ = "next free" integer, the "next free" integer is

increased by one, and the new mapping is remembered. If $(t_{j_{d,e}}, h_{j_{d,e}})$ has occurred previously in this iteration then it is mapped onto the same integer as previously.

Algorithm 3.1 implements the above process using a binary search tree to "remember" the mappings for each iteration. At the end of each iteration these mappings are "forgotten". During the j^{th} iteration, each distinct integer number ϵ_i in T_j "points" to the root of a binary search tree (see Aho et al. [1974] or Knuth [1973]) the nodes of which are labelled with the reals from H_j that have been found in combination with ϵ_i, together with the integers assigned to T_{j+1} for these combinations. Thus by searching the tree pointed to by ϵ_i we may quickly determine whether or not a given real value has previously occurred in combination with ϵ_i during this iteration. If it has occurred then the information in the tree will indicate what integer was assigned to T_{j+1} for that particular combination. If the combination has not occurred previously, then a new node is appended to the tree (in the appropriate position determined by the value of the real), together with the next available integer from the sequence 1,2,3,..... which will be assigned to T_{j+1} in the corresponding position.

The following variables and lists are used in Algorithm 3.1:

INTEGERPOINT -a list of length $n^2/2$ that is initialized to zero at the beginning of each iteration. During the j^{th} iteration if INTEGERPOINT(D), the D^{th} element in the list, is a non-zero number E, then it points to the root of the tree structure, TREE(E), for integer D occurring in T_j.

TREE -a list of length $n^2/2$ which contains the real numbers that have been encountered so far. If INTEGERPOINT(D)=W (where $W > 0$), then TREE(W) contains the first real number

found in combination with integer D during this iteration.

NEWTYP -a list of length $n^2/2$. The w^{th} entry, NEWTYP(W) contains the integer number that was assigned to the refined type matrix for the real in TREE(W).

LLINK -a list of length $n^2/2$. If a new node TREE(W) is appended to the tree structure, then LLINK(W) is set to zero. If LLINK(W)$\neq$0 then TREE(LLINK(W)) is the next real less than TREE(W) that was appended to the tree structure.

RLINK -a list of length $n^2/2$. If a new node TREE(W) is appended to the tree structure then RLINK(W) is set to zero. If RLINK(W)$\neq$0 then TREE(RLINK(W)) is the next real greater than TREE(W) that was appended to the tree.

FREESPACE -a variable which is initialized to 1 at the beginning of each iteration. It points to the next unused position in the list TREE.

NEXTINTEGER -a variable which is intialized to 1 at the beginning of each iteration. It points to the "next free" integer in the sequence 1,2,3,.....

Conceptually there are k refinements $T_2, T_3, \ldots\ldots, T_{k+1}$ of T_1, but in fact we need only one matrix T_1 which we overwrite.

T_1 -a nxn matrix. Upon input to the algorithm this matrix is irreducible and symmetric and contains only the consecutive integers from 1 through to z, where z is the total number of different integers contained in the matrix. Therefore if T_1 represents the adjacency matrix of a graph it contains only the integers 1 and 2 on input. Upon termination of the algorithm $T_\delta(A_1^*) = T_1$.

NUMTYP -a variable initialized to z, the number of different integers contained in matrix T_1 on input.

Conceptually there are k matrices H_1, H_2, ... ,H_k but since the algorithm processes only one H_i at a time, it is only necessary to reserve storage for one H_i matrix. In fact each element $h_{k_{i,j}}$ of a particular H_k may be determined directly by multiplying the appropriate row and column from matrix U_k (the matrix whose columns form a basis for the eigenvector space belonging to eigenvalue α_k), and so it is not necessary to reserve storage for any of the matrices H_1, H_2, ... , H_k since their entries may be calculated as needed.

On each invocation the algorithm repeats itself as many times as is necessary to achieve maximum refinement. Upon the first computation of $T_\delta(A_1^*)$ if the number of different integers in $T_\delta(A_1^*)$ equals the number of different integers occurring in the initial T_1 then the algorithm stops. On the other hand if $T_\delta(A_1^*)$ contains more different integers than the initial T_1, then $T_\delta(A_1^*)$ becomes the new T_1 and the type matrix of T_1 is recomputed.

Algorithm 3.1 Determination of the type matrix $T_\delta(A_1^*)$ of A_1 by direct iteration on the matrix decomposition

The algorithm assumes that A_1 is an irreducible symmetric matrix containing only the integers 1 through z where each integer in the sequence 1 through z occurs at least once in A_1.

Step 1: Initialize type matrix T_1 = the input matrix A_1.
NUMTYP = z the number of different integers in T_1.

Step 2: Initialize variable which will keep track of which H_i matrix is being processed. Each H_i is computed from the matrix decomposition of T_1. W:=1.

Step 3: (Processing lower triangle of H_w and current T_1)
Initialize pointers.
FOR I:=1 UNTIL (N**2)/2 DO INTEGERPOINT(I):=0.
FREESPACE:=NEXTINTEGER:=1.
I:=1.

Step 4: J:=1.

Step 5: Let β:=$h_{w_{i,j}}$ the real number in the i,jth position of H_w. Pick up the pointer for the corresponding entry in T_1 and if it is non-zero search for β in the tree structure.
κ:=INTEGERPOINT($t_{1_{i,j}}$).
IF κ>0 THEN step 7.

Step 6: This is the first occurrence of real β from H_w with integer $t_{1_{i,j}}$ from T_1. Set up a binary tree for this integer.
κ:=FREESPACE
FREESPACE:=FREESPACE+1
INTEGERPOINT($t_{1_{i,j}}$):= κ
TREE(κ):= β
NEWTYP(κ):=NEXTINTEGER
NEXTINTEGER:=NEXTINTEGER+1
LLINK(κ):=RLINK(κ):=0
Go to step 8.

Step 7: Look for β in the tree structure of integer $t_{1_{i,j}}$ and if it is not there append it in the appropriate place.

```
WHILE β ≠ TREE(κ) DO
  BEGIN
  FLAG:=TRUE
  IF β> TREE(κ) THEN L:=RLINK(κ) ELSE
                    BEGIN L:=LLINK(κ)  FLAG:=FALSE END.
  IF L ≠ 0 THEN κ:=L ELSE
     BEGIN
     IF FLAG THEN RLINK(κ):= FREESPACE ELSE LLINK(κ):=FREESPACE
     κ:=FREESPACE
     FREESPACE:=FREESPACE+1
     TREE(κ):= β
     NEWTYP(κ):=NEXTINTEGER
     NEXTINTEGER:=NEXTINTEGER+1
     LLINK(κ):=RLINK(κ):=0
     END.
  END.
```

Step 8: Assign a new integer in the i,j^{th} position of T_1.
$t_{1\,i,j} := NEWTYP(K)$

Step 9: Process the next element in the i^{th} row of T_1.
J:=J+1
IF $J \leq I$ THEN Step 5.

Step 10: Process the next row in the lower triangle of T_1.
I:=I+1
IF $I \leq N$ THEN Step 4

Step 11: Process the next H_w.
W:=W+1
IF $W \leq K$ THEN Step 3.

Step 12: If the number of integers in T_1 has increased from the initial T_1 then repeat the whole process again.
Copy the lower triangle of T_1 into the upper triangle of T_1.
IF NEXTINTEGER-1 > NUMTYP THEN let NUMTYP:=NEXTINTERGER-1 and GO TO step 2.

Step 13: T_1 is the nxn symmetric type matrix $T_\delta(A_1^*)$ of the original input matrix A_1.
TERMINATE.

The order of the computation for this algorithm is discussed in Chapter 5 in connection with the backtracking isomorphism algorithm. In the isomorphism algorithm, the above algorithm is used to simultaneously type two given matrices A_1 and A_2. The method is exactly the same except that on each iteration we form two type matrices T_1 and T_1' belonging to A_1 and A_2 respectively. On any one of the k iterations, the same mappings are used to transform T_1' by examination of H_i' as were used to transform T_1 by examination of H_i. If at any stage T_1 and T_1' do not contain the same number of different integers then the algorithm stops because this is an indication that the input matrices A_1 and A_2 are not isomorphic.

3.2 Properties of the Type Matrix $T_\delta(A_1^*)$

The following is a development of results which provides us with a means of exploiting the type matrix of a matrix A_1 either to determine the existence of an isomorphism with another matrix A_2 (discussed in section 3.3) or to find the generators of the automorphism group of A_1. Note that by definition the formation of k-vector type matrices and type matrices of two given matrices only has a meaning if the matrices have the same eigenvalues of the same multiplicities.

Theorem 3.2.1

$A_1 \cong A_2$ by isomorphism σ represented by the permutation matrix P iff $A_1^* = P^T A_2^* P$ where A_1^* and A_2^* are the k-vector type matrices of A_1 and A_2 respectively.

Proof:

(i) Assume $A_1 \cong A_2$ by isomorphism σ represented by the permutation matrix P where $A_1 = P^T A_2 P$.
Then by Theorem 2.3.4 $H_i = P^T H_i' P$ for all i=1,2,...,k.
Hence $A_1^* = P^T A_2^* P$ since A_1^* and A_2^* are formed by the concatenation of entries from H_i and H_i' for i=1,2,....,k.

(ii) Assume that $A_1^* = P^T A_2^* P$.
Since A_1^* and A_2^* are formed by the concatenation of entries from H_i and H_i' respectively, then $H_i = P^T H_i' P$ for all i=1,2,...,k.
Therefore by Theorem 2.3.4 $A_1 \cong A_2$ by isomorphism σ represented by permutation matrix P.

QED.

Theorem 3.2.2

$A_1 \cong A_2$ by an isomorphism σ represented by the permutation matrix P iff $T_\delta(A_1^*) = P^T T_\delta(A_2^*) P$.

Proof:

(i) Assume $A_1 \cong A_2$ by isomorphism σ represented by permutation matrix P.
Then by Theorem 3.2.1 we have: $A_1^* = P^T A_2^* P$ which implies that $T_\delta(A_1^*) = P^T T_\delta(A_2^*) P$ since δ induces a mapping from A_1^* to $T_\delta(A_1^*)$ and from A_2^* to $T_\delta(A_2^*)$.

(ii) Assume there exists a permutation matrix P such that $T_\delta(A_1^*) = P^T T_\delta(A_2^*)P$.
Since δ maps different k-tuples onto different integers, this induces an inverse mapping from $T_\delta(A_1^*)$ to A_1^* and from $T_\delta(A_2^*)$ to A_2^*.
Hence $A_1^* = P^T A_2^* P$ and so by Theorem 3.2.1 $A_1 \cong A_2$ by isomorphism σ represented by permutation matrix P.

QED.

Corollary 3.2.3

If $T_\delta(A_1^*)$ and $T_\delta(A_2^*)$ are not cospectral then $A_1 \ncong A_2$.

Proof:

This follows directly from Theorem 3.2.2.

Theorem 3.2.4

σ is an automorphism of Γ_{A_1} (the automorphism group of the matrix A_1) iff $\sigma \in \Gamma_{T_\delta(A_1^*)}$.

Proof:

(i) Assume $\sigma \in \Gamma_{A_1}$.
By the definition of automorphism this implies that there exists a permutation matrix P representing σ such that $A_1 = P^T A_1 P$.
Hence by Theorem 3.2.2 $T_\delta(A_1^*) = P^T T_\delta(A^*_1)P$. Which is to say that $\sigma \in \Gamma_{T_\delta(A_1^*)}$.

(ii) Assume $\sigma \in \Gamma_{T_\delta(A_1^*)}$.
By the definition of automorphism of a matrix there exists a permutation matrix P representing σ such that $T_\delta(A_1^*) = P^T T_\delta(A_1^*)P$.
Hence by Theorem 3.2.2 P represents an automorphism σ of A_1.

QED.

Suppose we have two isomorphic graphs G_1 and G_2 with an isomorphism σ mapping $G_2 \rightarrow G_1$. Then σ may be represented by a permutation matrix P such that $A_1 = P^T A_2 P$. It is well known that the effect of the multiplication on the right hand side of this expression is to transform A_2 into A_1 by permuting the appropriate rows and columns of A_2 according to the

mapping σ. In particular, if σ maps i onto j then the matrix multiplication $P^T A_2 P$ would physically move the i^{th} diagonal element of A_2 into the position of the j^{th} diagonal element in A_1. To make this absolutely clear we demonstrate with the following example:

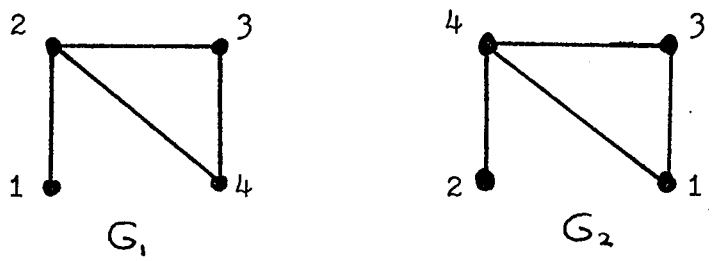


Figure 3.2.1 Two isomorphic graphs.

Consider the isomorphic graphs shown in Figure 3.2.1. An isomorphism from G_2 to G_1 is $\sigma = (1\ 4\ 2)(3)$ which is represented by the permutation matrix $P = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$. The adjacency matrix $A_2 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$. If the diagonal elements are "marked" with the numbers 2, 3, 4, and 5 in order to distinguish them one from the other then $A_2 = \begin{bmatrix} 2 & 0 & 1 & 1 \\ 0 & 3 & 0 & 1 \\ 1 & 0 & 4 & 1 \\ 1 & 1 & 1 & 5 \end{bmatrix}$. Remembering that the numbers 2, 3, 4, and 5 are in place of the 0 diagonal elements, then upon multiplying $P^T A_2 P$ we get: $P^T A_2 P = \begin{bmatrix} 3 & 1 & 0 & 0 \\ 1 & 5 & 1 & 1 \\ 0 & 1 & 4 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}$ which is just the matrix A_1 with the diagonal 0's replaced by the numbers 3, 5, 4, and 2. Hence the diagonal elements from A_2 have been physically moved onto the appropriate diagonal elements from A_1 according to the mapping σ.

The above observation allows us to prove the following results:

Theorem 3.2.5

Let $A_1 \cong A_2$. Let σ be any isomorphism that maps A_2 onto A_1 and suppose that σ maps i onto j where $i,j \in \{1,2,\ldots,n\}$. Let P be the permutation matrix representing σ. Suppose we transform A_1 and A_2 to A_1' and A_2' by setting the i^{th} diagonal element of A_2 and the j^{th} diagonal element of A_1 equal to γ (where γ is a positive integer distinct from any other integer occurring in A_1 and A_2). Then the only isomorphisms between A_1' and A_2' are those isomorphisms of A_1 and A_2 which map i onto j.

Proof:

Let $A_1 = P^T A_2 P$ where P represents an isomorphism σ and suppose that σ maps i onto j. Since σ maps i onto j, then the multiplication $P^T A_2 P$ physically moves the i^{th} diagonal element of A_2 onto the j^{th} diagonal element of A_1. Therefore if P is an isomorphism of A_1 and A_2, then $a_{2_{i,i}} = a_{1_{j,j}}$. Therefore by altering A_1 and A_2 by setting $a_{2_{i,i}} = a_{1_{j,j}} = \gamma$ (where γ is an integer distinct from the other diagonal elements of A_1 and A_2) we do not change the effect of this multiplication. This implies that P is also an isomorphism of A_1' and A_2'. Furthermore, it is obvious that since the i^{th} diagonal element of A_2' and j^{th} diagonal element of A_1' are distinct from the others, then any isomorphism between A_2' and A_1' must map i onto j. Finally it remains to show that any isomorphism of A_1' and A_2' is an isomorphism of A_1 and A_2.

Suppose $P_1^T A_2' P_1 = A_1'$. We know from above that P_1 must map i onto j. This implies that the i^{th} diagonal element of A_2' and the j^{th} diagonal element of A_1' are equal. Therefore if we change A_1' and A_2' back to A_1 and A_2 by setting $a_{2_{i,i}} = a_{1_{j,j}}$ = their original value, then the multiplication $P_1^T A_2 P_1 = A_1$ would still be valid.

QED.

Corollary 3.2.6

Suppose we alter two given matrices A_1 and A_2 as in the above theorem by setting $a_{1_{j,j}} = a_{2_{i,i}} = \gamma$ where γ is distinct from the other diagonal elements. Then if A_1' and A_2' are not cospectral there exists no isomorphism between A_2 and A_1 that maps i onto j.

Theorem 3.2.7

$A_1 \cong A_2$ by an isomorphism σ mapping i onto j iff $T_\delta(A_1'^*) = P^T T_\delta(A_2'^*)P$ where P is the permutation matrix representing σ, and A_1' & A_2' are formed from A_1 and A_2 upon setting $a_{2_{i,i}} = a_{1_{j,j}} = \gamma$, where γ is any positive integer distinct from the others occurring in A_1 and A_2.

Proof:

(i) Assume $A_1 \cong A_2$ by an isomorphism σ mapping i onto j where P is a permutation matrix representing σ.
Theorem 3.2.5 implies that $A_1' = P^T A_2' P$ where A_1' and A_2' are formed as above.
Hence by Theorem 3.2.2 $T_\delta(A_1'^*) = P^T T_\delta(A_2'^*)P$.

(ii) Assume $T_\delta(A_1'^*) = P^T T_\delta(A_2'^*)P$ where P is a permutation matrix representing a mapping σ, and maps i onto j. A_1' and A_2' are formed as above.
By Theorem 3.2.2 we have $A_1' = P^T A_2' P$, and therefore by Theorem 3.2.5 $A_1 \cong A_2$ by isomorphism σ which maps i onto j.

QED.

Corollary 3.2.8

Suppose we alter two given matrices A_1 and A_2 to A_1' and A_2' as in the above theorem. If $T_\delta(A_1'^*)$ and $T_\delta(A_2'^*)$ are not cospectral, then there is no isomorphism between A_2 and A_1 that maps i onto j.

3.3 Determining the Existence of an Isomorphism

In this section we demonstrate how the theorems of section 3.2 may be used to search for an isomorphism between two cospectral graphs whose adjacency matrices have a similar matrix decomposition. An informal description of the backtracking process is given followed by an example. Finally a detailed algorithm for the backtracking procedure is presented.

Consider two cospectral matrices A_1 and A_2 that have a similar matrix decomposition. Since this is the case, it is impossible to detect the non-isomorphism of these matrices using the methods of Chapter 2. For this situation, we propose a backtracking algorithm which will either find an isomorphism or disprove isomorphism. The central idea of our backtracking process is that at any given stage we assume that the matrices currently being examined are isomorphic by some permutation matrix P. There may be further assumptions about P, such as: P must map $i \rightarrow j$ for example (this would mean that the i^{th} diagonal element from the first matrix and the j^{th} diagonal element from the second matrix are equal to each other and unique within the matrix in which they occur). If all the diagonal elements of each current matrix are distinct then P is completely determined as it would have to map a diagonal element from the first matrix onto a diagonal element with the same value in the second matrix. If at least one diagonal element is repeated from each matrix then the algorithm attempts to "increase" its knowledge of P by altering the current matrices (as in Theorem 3.2.5) in order to restrict the possibilities for P even more.

Consider the two matrices A_1 and A_2 shown in Figure 3.3.1:

$A_1 =$

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

$A_2 =$

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

Figure 3.3.1 Two 10x10 symmetric matrices.

If there exists a permutation matrix P such that $A_1 = P^T A_2 P$ then it is obvious that P must map $1 \rightarrow 1$ since $a_{1_{1,1}} = a_{2_{1,1}} = 1$ are unique. In this case no other diagonal elements are unique. If A_1 and A_2 are transformed to A_1' and A_2' upon setting $a_{1_{2,2}} = a_{2_{3,3}} = 12$ (since 11 is the highest integer occurring in the matrices) then by Theorems 3.2.5 and 3.2.7 we have $A_1' = P^T A_2' P$ and $T_\delta(A_1'^*) = P^T T_\delta(A_2'^*)P$ provided that $2 \rightarrow 3$ is a valid additional restriction on P. If either the matrices A_1' and A_2' or $T_\delta(A_1'^*)$ and $T_\delta(A_2'^*)$ are not cospectral then we conclude that the most recent restriction on P is not valid. Similarily if the matrices $T_\delta(A_1'^*)$ and $T_\delta(A_2'^*)$ do not have the same set of diagonal elements then the most recent restriction on P is not valid. If the most recent restriction on P

fails, then the algorithm "backs-up" and tries another alternative. If $2 \rightarrow 3$ fails for the matrices in Figure 3.3.1 then the procedure would try $2 \rightarrow 6$. If the most recent restriction on P does not fail then the matrices $T_{\delta}(A_1'^*)$ and $T_{\delta}(A_2'^*)$ become the new "current" matrices and the above process of looking for a further restriction on P is repeated. The computation continues until either all the possibilities are exhausted or a "complete" permutation P is discovered. Note that the "tree" of possibilities may be pruned quickly because it is not necessary to form a complete mapping to discover that a given partial mapping is invalid.

As an illustration of the process, consider the two isomorphic Petersen graphs shown in Figure 3.3.2:

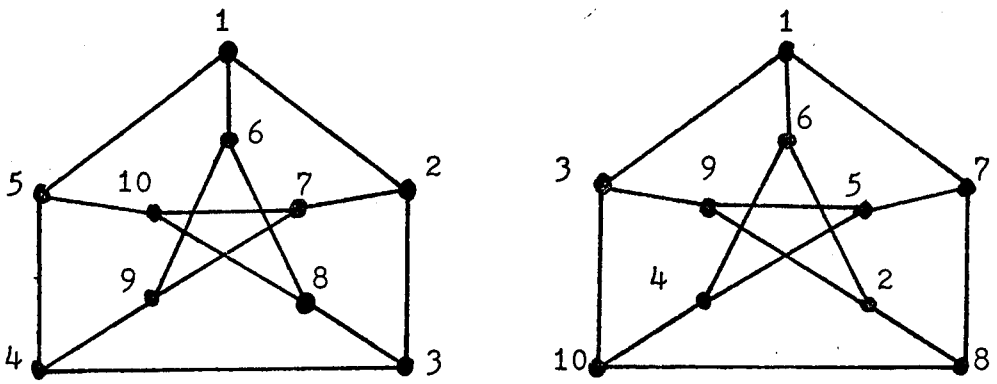


Figure 3.3.2 Two isomorphic Petersen graphs.

Since the methods of Chapter 2 are unable to detect these graphs as being non-isomorphic, the type matrices $T_{\delta}(A_1^*)$ and $T_{\delta}(A_2^*)$ (shown in Figure 3.3.3) are computed. By Theorem 3.2.2 if $A_1 = P^T A_2 P$ then $T_{\delta}(A_1^*) = P^T T_{\delta}(A_2^*) P$. The first step in finding out more about P is to alter the matrices A_1 and A_2 to A_1' and A_2' by setting $a_{1_{1,1}} = a_{2_{1,1}} = 4$. Next the type matrices $T_{\delta}(A_1'^*)$ and $T_{\delta}(A_2'^*)$ are determined as shown in Figure 3.3.4:

$T_{\delta}(A_1^*)=$

1	2	3	3	2	2	3	3	3	3
2	1	2	3	3	3	2	3	3	3
3	2	1	2	3	3	3	2	3	3
3	3	2	1	2	3	3	3	2	3
2	3	3	2	1	3	3	3	3	2
2	3	3	3	3	1	3	2	2	3
3	2	3	3	3	3	1	3	2	2
3	3	2	3	3	2	3	1	3	2
3	3	3	2	3	2	2	3	1	3
3	3	3	3	2	3	2	2	3	1

$T_{\delta}(A_2^*)=$

1	3	2	3	3	2	2	3	3	3
3	1	3	3	3	2	3	2	2	3
2	3	1	3	3	3	3	3	2	2
3	3	3	1	2	2	3	3	3	2
3	3	3	2	1	3	2	3	2	3
2	2	3	2	3	1	3	3	3	3
2	3	3	3	2	3	1	2	3	3
3	2	3	3	3	3	2	1	3	2
3	2	2	3	2	3	3	3	1	3
3	3	2	2	3	3	3	2	3	1

Figure 3.3.3 The type matrices of two isomorphic Petersen graphs.

$T_{\delta}(A_1'^*)=$

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

$T_{\delta}(A_2'^*)=$

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

Figure 3.3.4 The type matrices fixing $1 \rightarrow 1$.

The matrices A_1' and A_2' prove to be cospectral and their type matrices have the same set of diagonal elements. At this stage we assume that the most recent restriction on P (namely 1→1) is valid and proceed to further restrict P. Set $A_1 = T_\delta(A_1'^*)$ and $A_2 = T_\delta(A_2'^*)$. The next diagonal element which is not unique is $a_{1_{2,2}} = 3$, and $a_{2_{3,3}} = 3$ is the first equal diagonal element in A_2. A_1' and A_2' are formed by setting $a_{1_{2,2}} = a_{2_{3,3}} = 12$. The type matrices $T_\delta(A_1'^*)$ and $T_\delta(A_2'^*)$ are computed and given in Figure 3.3.5:

$T_\delta(A_1'^*) =$

1	2	4	7	11	11	4	7	7	7
2	3	5	8	12	12	5	8	8	8
4	5	6	9	13	13	18	9	19	19
7	8	9	10	14	16	19	20	21	22
11	12	13	14	15	17	13	16	16	14
11	12	13	16	17	15	13	14	14	16
4	5	18	19	13	13	6	19	9	9
7	8	9	20	16	14	19	10	22	21
7	8	19	21	16	14	9	22	10	20
7	8	19	22	14	16	9	21	20	10

$T_\delta(A_2'^*) =$

1	7	2	7	7	11	11	7	4	4
7	10	8	22	20	14	16	21	9	19
2	8	3	8	8	12	12	8	5	5
7	22	8	10	21	14	16	20	19	9
7	20	8	21	10	16	14	22	9	19
11	14	12	14	16	15	17	16	13	13
11	16	12	16	14	17	15	14	13	13
7	21	8	20	22	16	14	10	19	9
4	9	5	19	9	13	13	19	6	18
4	19	5	9	19	13	13	9	18	6

Figure 3.3.5 Type matrices fixing 1→1 and 2→3.

This step succeeds in that A_1' and A_2' are cospectral and their type matrices have the same diagonal elements. Repeating the process, set $a_{1_{3,3}} = a_{2_{9,9}} = 23$. The resulting type matrices $T_\delta(A_1'^*)$ and $T_\delta(A_2'^*)$ are given in Figure 3.3.6:

$T_\delta(A_1'^*)=$

1	2	4	7	11	11	18	7	25	25
2	3	5	8	12	12	19	8	26	26
4	5	6	9	13	13	20	9	27	27
7	8	9	10	14	16	21	24	28	32
11	12	13	14	15	17	22	16	29	30
11	12	13	16	17	15	22	14	30	29
18	19	20	21	22	22	23	21	31	31
7	8	9	24	16	14	21	10	32	28
25	26	27	28	29	30	31	32	33	34
25	26	27	32	30	29	31	28	34	33

$T_\delta(A_2'^*)=$

1	7	2	25	7	11	11	25	4	18
7	10	8	32	24	14	16	28	9	21
2	8	3	26	8	12	12	26	5	19
25	32	26	33	28	30	29	34	27	31
7	24	8	28	10	16	14	32	9	21
11	14	12	30	16	15	17	29	13	22
11	16	12	29	14	17	15	30	13	22
25	28	26	34	32	29	30	33	27	31
4	9	5	27	9	13	13	27	6	20
18	21	19	31	21	22	22	31	20	23

Figure 3.3.6 Type matrices fixing 1→1, 2→3, and 3→9.

The diagonal elements are not all unique yet so we set $a_{1_{9,9}} = a_{2_{4,4}} = 35$. The resulting type matrices are shown in Figure 3.3.7:

$T_\delta(A_1'^*)=$

1	2	4	7	11	16	22	29	37	46
2	3	5	8	12	17	23	30	38	47
4	5	6	9	13	18	24	31	39	48
7	8	9	10	14	19	25	32	40	49
11	12	13	14	15	20	26	33	41	50
16	17	18	19	20	21	27	34	42	51
22	23	24	25	26	27	28	35	43	52
29	30	31	32	33	34	35	36	44	53
37	38	39	40	41	42	43	44	45	54
46	47	48	49	50	51	52	53	54	55

$T_\delta(A_2'^*)=$

1	29	2	37	7	16	11	46	4	22
29	36	30	44	32	34	33	53	31	35
2	30	3	38	8	17	12	47	5	23
37	44	38	45	40	42	41	54	39	43
7	32	8	40	10	19	14	49	9	25
16	34	17	42	19	21	20	51	18	27
11	33	12	41	14	20	15	50	13	26
46	53	47	54	49	51	50	55	48	52
4	31	5	39	9	18	13	48	6	24
22	35	23	43	25	27	26	52	24	28

Figure 3.3.7 Type matrices fixing 1→1, 2→3, 3→9, and 9→4.

At this stage since the matrices A_1' and A_2' are cospectral and the diagonal elements of their type matrices are unique, we have determined a complete permutation (1) (2 3 9 4 5 7 10 8) (6). The multiplication with the original adjacency matrix A_2 confirms that this is in fact an isomorphism. In this thesis it is considered that finding one isomorphism is sufficient. However, it is easily seen that all the isomorphisms may be determined by using this same technique to backtrack through all the various possibilities.

The basic idea behind the backtracking algorithm is to form "partial" mappings between A_1 and A_2 based on mapping diagonal elements of the same type onto each other. At any stage if the partial mapping causes the resultant matrices to be non-cospectral or their type matrices to differ along the diagonal then the most recent piece of mapping is discarded and the next possibility is tried. When a complete mapping is formed (ie. two type matrices with unique diagonal elements) then this is tested by the matrix multiplication $P^T A_1 P$ to verify that it is in fact an isomorphism. If the mapping is not an isomorphism then the algorithm "backs-up" and tries the next possibility. This process continues until either an isomorphism is found or all the possibilities are exhausted in which case the graphs are not isomorphic.

The examples that were tried on the computer indicate that the tree is pruned near the root when it begins to form partial mappings for non-isomorphic graphs. The bipartite graphs in Figure 3.0.1 , for example, required 10 backtracks at the top level to disprove isomorphism. Node 1 from the first graph may possibly map onto any one of nodes 1 through 10 from the second graph. However, each time a partial mapping $1 \rightarrow r$ where $r \in \{1,2,..,10\}$ was formed, the algorithm rejected this as an impossible mapping. So in this case the search tree of possibilities was pruned close to the root. In fact, as yet, we have been unable to

find a pair of non-isomorphic cospectral graphs which require more than n backtracks at the top level to disprove isomorphism. The algorithm performs equally well for isomorphic graphs. For many graphs (including all those in Chapter 2) the algorithm successfully determines isomorphism using 0 backtracks. A timing study of the algorithm's performance is given in Chapter 5 along with a predicted bound for the computation time.

Algorithm 3.3.1 Backtracking to search for an isomorphism.

At this point it is assumed that the methods of Chapter 2 failed to detect the non-isomorphism of A_1 and A_2.

Step 1: For the purpose of backtracking set the current level counter LEV:=1.

Step 2: Compute the type matrices $T_\delta(A_1^*)$ and $T_\delta(A_2^*)$ using Algorithm 3.1. Let W:= the highest integer number in the type matrices. If the type matrices are not cospectral then $A_1 \not\cong A_2$ by Corollary 3.2.3.

Step 3: Heapsort the diagonal elements of $T_\delta(A_1^*)$ into ascending order remembering which columns they came from. Similarily sort the elements of $T_\delta(A_2^*)$. If there is not the same set of diagonal elements in each type matrix, then if LEV $\neq$ 1 go to step 7 else quit, as the matrices are not isomorphic.

Step 4: If every diagonal element is not unique then step 5; otherwise we have a complete mapping between the two matrices. If this mapping is an isomorphism then TERMINATE. Otherwise restore the previous type matrix and try the next mapping at the current level via step 7.

Step 5: Increment the current level counter LEV:=LEV+1.
Let X:=MAP(LEV-1,2) (this is the column of the second matrix that was mapped onto at the previous level. If 3→4 was the previous choice then X:=4.) If diagonal element X is unique in $T_\delta(A_1^*)$ or if this is level 1 then set X:= the column number of the first diagonal element in the smallest group of non-repeating ones in $T_\delta(A_1^*)$.

Step 6: Let diagonal element X have value γ_{LEV} and suppose that it repeats $P_{LEV} \leq n$ times. Let $\{a_{LEV,1}, a_{LEV,2}, \ldots, a_{LEV,P_{LEV}}\}$ be the diagonal elements of $T_{\delta}(A_2^*)$ having value γ_{LEV}. Save these numbers in a level list LEVLIST(LEV,1) through to LEVLIST(LEV,P_{LEV}). Initialize the level pointer LEVPT(LEV):=0. Begin the partial mapping for this level by setting MAP(LEV,1):=X (where MAP(LEV,2) will contain the column number of $T_{\delta}(A_2^*)$ which X maps onto).

Step 7: Form the next mapping for the current level. Increment the level pointer LEVPT(LEV):=LEVPT(LEV)+1. IF LEVPT(LEV) $>$ P_{LEV} THEN the current level is exhausted so back-up a level by setting LEV:=LEV-1, restoring the type matrices, and restarting step 7. If Level 1 is exhausted then the graphs are not isomorphic and so the algorithm TERMINATES.) Otherwise set MAP(LEV,2):=LEVLIST(LEV,LEVPT(LEV)) (because MAP(LEV,1) $\rightarrow$ MAP(LEV,2)).

Step 8: Let X:=MAP(LEV,1). Let Y:=MAP(LEV,2). Set the X^{th} diagonal element of $T_{\delta}(A_1^*)$ equal to W+1. Similarily set the Y^{th} diagonal element of $T_{\delta}(A_2^*)$ equal to W+1. If the modified matrices $T_{\delta}(A_1^*)'$ and $T_{\delta}(A_2^*)'$ are not cospectral then the most recent partial mapping is invalid and so get the next one by going to step 7.

Step 9: Set $A_1 := T_{\delta}(A_1^*)'$ and $A_2 := T_{\delta}(A_2^*)'$. Compute the type matrices $T_{\delta}(A_1^*)$ and $T_{\delta}(A_2^*)$ using Algorithm 3.1. Set W:= the highest integer in these type matrices. If these type matrices are not cospectral then get the next possibility for this level via step 7; otherwise go to step 3.

To economize on memory in the implementation of this backtracking algorithm it is not necessary to "remember" the type matrix for each level in case of the necessity to backup the tree of possibilities. All that is needed to form the type matrix at a given level is the type matrix that the algorithm originally started with along with a list of the partial mappings used to reach that level (the list MAP in the algorithm contains this information).

CHAPTER 4

Finding a Set of Generators for the Automorphism Group

4.0 A Step by Step Example

In this Chapter we present a modification of the backtracking isomorphism algorithm that enables a determination of the order of the automorphism group together with a small set of generators which generate the whole group. In the course of a search for a set of generators, the algorithm utilizes the algebraic properties of the subgroups of the automorphism group while actually finding the automorphisms by "fixing" nodes of two identical copies of the type matrix $T_\delta(A_1^*)$ of the graph G_1 as in Algorithm 3.3.1. For finding the automorphism group of G_1 is clearly equivalent to the problem of finding all the isomorphisms between two identical copies of the type matrix $T_\delta(A_1^*)$.

Before describing the process in any detail it is necessary to state some terminology and results from the Theory of Groups taken from Van der Waerden [1970]:

Let $\{a,b,.....\}$ be a set of arbitrary elements of a group Γ. Then the group generated by $\{a,b,.....\}$ consists of all products each of a finite number of these elements and their inverses; furthermore $\{a,b,....\}$ are said to be generators of this group. A complex is defined as an arbitrary set of elements of a group Γ. By the product $C_1.C_2$ of two complexes C_1 and C_2 we understand the set of all products $c_1.c_2$ where $c_1 \in C_1$ and $c_2 \in C_2$. If S is a subgroup of a group Γ and $a \in \Gamma$ then the complex a.S is called a left coset of S in Γ. Similarily S.a is called a right coset of S in Γ. The number of different cosets of a subgroup S in Γ is called the index of S in Γ.

Result 4.0.1

The cosets of a subgroup S in Γ constitute a partition of the group Γ.

Result 4.0.2

If N is the order of Γ (assumed finite), n the order of S and j the index of S in Γ then N = jn.

Let Γ be a group of permutations on a set X. Then the stabilizer S_x of an element $x \in X$ is defined to be the subgroup of Γ which keeps x fixed. Obviously the index of S_x in Γ is equal to the length of the orbit of x in the automorphism partitioning of Γ.

In our automorphism algorithm both the concepts of a coset of a subgroup in a group and of the stabilizer subgroup of an element in a group play an important role in the search strategy for a set of generators of the automorphism group. By the definition of coset it is evident that if $S_{a_{11}}$ is the stabilizer subgroup of element a_{11} in Γ_{G_1}, where the index of $S_{a_{11}}$ in Γ_{G_1} is w, then there exist w-1 different permutations $\{d_2, d_3, \ldots\ldots, d_w\}$ in Γ_{G_1} with the property that $\Gamma_{G_1} = S_{a_{11}} + S_{a_{11}}.d_2 + S_{a_{11}}.d_3 + \ldots\ldots + S_{a_{11}}.d_w$. If the generators of the subgroup $S_{a_{11}}$ are known then in order to determine the generators of the whole automorphism group the algorithm must find w-1 elements of Γ_{G_1} that will generate the other cosets of $S_{a_{11}}$ in the above partition of Γ_{G_1}. It should be noted here that in practice the algorithm searches for a permutation d_i with large orbits so that quite often the powers of d_i will include some of the other possibilities in $\{d_2, d_3, \ldots, d_w\}$.

The core of the automorphism algorithm is a recursive process because the same principles as described above for Γ_{G_1} are first employed to search for the generators of $S_{a_{11}}$. Specifically, we first determine the generators of $S_{a_{21}}$ the stabilizer of a_{21} in $S_{a_{11}}$ and the elements of $S_{a_{11}}$ that will produce the different cosets of $S_{a_{21}}$ in $S_{a_{11}}$.

For if the index of $S_{a_{21}}$ in $S_{a_{11}}$ is z then there exist z-1 elements $\{f_2, f_3, \ldots\ldots, f_z\}$ in $S_{a_{11}}$ with the property that $S_{a_{11}} = S_{a_{21}} + S_{a_{21}}.f_2 + S_{a_{21}}.f_3 + \ldots\ldots + S_{a_{21}}.f_z$. In order to determine the generators of $S_{a_{21}}$ the algorithm searches for a stabilizer subgroup of $S_{a_{21}}$. This process repeats itself r times until the largest stabilizer subgroup contained in $S_{a_{r1}}$ is I, the identity permutation. Once the bottom level generators are determined the process climbs back up the chain of nested possibilities finding generators of all the different cosets at each level. P. Gibbons [1976] uses a similar algebraic approach to generating the automorphism group; however, he uses a different technique to determine the possibilities and then to find the automorphisms.

In practice the automorphism algorithm begins by forming the type matrix $T_{\delta}(A_1^*)$ of G_1 in order to obtain an initial partition of the nodes. Recall that the position numbers of the elements with the same integer value along the main diagonal may possibly map onto each other in Γ_{G_1} and so are candidates for the nodes of a transitive subgraph. Certainly by Theorem 3.2.2 diagonal elements with a different integer value are not mapped onto each other by an automorphism of Γ_{G_1}. Let $\{a_{11}, a_{12}, \ldots\ldots, a_{1b}\}$ be an ascending list of integers that are the position numbers of the smallest group of repeating diagonal elements in $T_{\delta}(A_1^*)$. If there are no repeating diagonal elements at this stage then the graph is rigid. Although these nodes are candidates for a transitive subgraph it is not necessarily the case as will be seen shortly in the step by step example.

The first subgoal of the algorithm is to investigate $S_{a_{11}}$ and to determine the index of $S_{a_{11}}$ in Γ_{G_1}. A further subgoal in order to achieve this goal is to find a stabilizer subgroup $S_{a_{21}}$ of a_{21} in $S_{a_{11}}$. More generally these stabilizer subgroups are nested as shown in Figure 4.0.1:

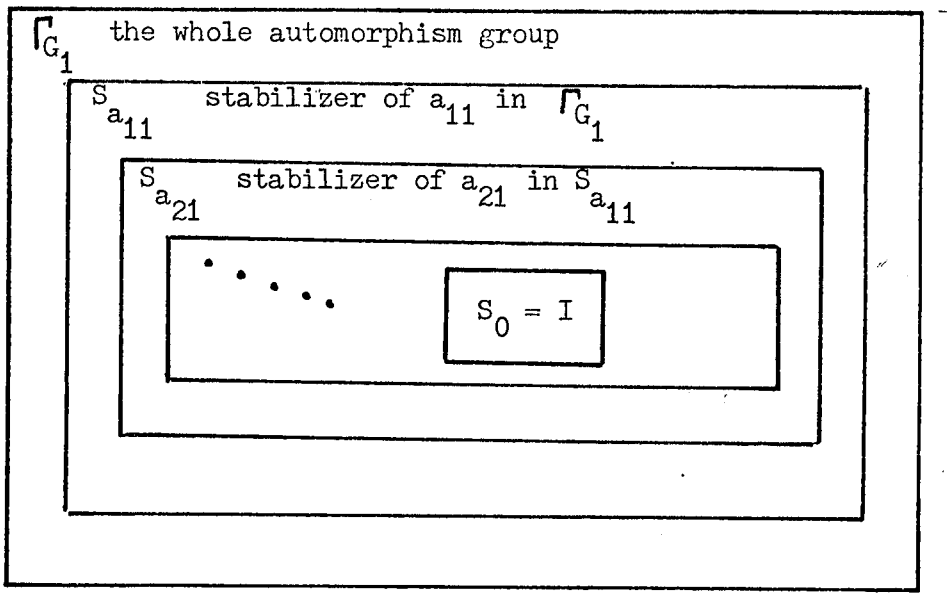


Figure 4.0.1 Chain of nested stabilizer subgroups of Γ_{G_1}.

At the bottom of the chain of nested stabilizer subgroups is the identity permutation I which fixes all n nodes of G_1. Clearly if $S_{a_{11}} = I$ then either the graph is rigid or there are at most n automorphisms.

The first step in the process is to "fix" node a_{11} by setting the a_{11}^{th} diagonal element of $T_\delta(A_1^*)$ equal to a unique integer and finding the type of this modified matrix. The diagonal elements of this new type matrix indicate the possibilities for $S_{a_{21}}$. If all the diagonal elements are not unique then let $\{a_{21}, a_{22}, \ldots\ldots, a_{2q}\}$ in ascending order be the integer position numbers of the smallest set of repeating diagonal elements (within the range of the ones we started with at the top level). At this point the a_{21}^{th} diagonal element is fixed and the process repeats itself. A formal statement of the algorithm will be presented in section 4.1. In the meantime it is hoped that the stepwise analysis of the following example will serve to illustrate the principal strategies of the algorithm.

Consider the 25-point graph (ii) given in Figure 3.0.1. This is a bipartite graph of a design and recall that it is one of the pair of graphs found by Mathon to be a counter-example to Corneil's automorphism conjecture. The initial type matrix of this graph is shown in

Figure 4.0.2. Not surprisingly the first ten columns (nodes) are distinguished from the other fifteen columns (nodes) by the numbers along the main diagonal. Consequently our initial partition of the nodes into subsets of candidates for transitive subgraphs is $\{1,2,3,4,5,6,7,8,9,10\}$ and $\{11,12,13,14,15,16,17,18,19,20,21,22,23,24,25\}$.

In order to discern the possible subgroup structures of S_1 the stabilizer of node 1 in Γ_{G_1}, the algorithm will fix a series of diagonal elements and see what choices are available at each stage. Initially we manipulate one copy of the type matrix $T_{\mathcal{S}}(A_1^*)$. So at the top level diagonal element 1 may possibly map onto any one of $\{1,2,3,4,5,6,7,8,9,10\}$. The first step in exploring the structure of S_1 is to fix node 1 by setting the first diagonal element of the type matrix to the value 8. The type matrix of this modified matrix is computed and this becomes the new current type matrix for the next level. Observe in Figure 4.0.3 that the diagonal elements are still not all unique after fixing $1\rightarrow 1$ and so there is some further choice. The second diagonal element is the smallest non-unique one. At this stage diagonal element 2 from the type matrix has the same integer value as diagonal elements $\{2,5,6,10\}$ and so $2\rightarrow\{2,5,6,10\}$ are the only possibilities at this level.

Now in order to investigate the structure of subgroup S_2 in S_1 we "fix" the second diagonal element in the current type matrix equal to an integer distinct from the rest. The type matrix of this modified matrix becomes the new current type matrix at the next level (shown in Figure 4.0.4). Now with 1 and 2 fixed the third diagonal element is the smallest non-unique one. The choice at this stage is $3\rightarrow\{3,8\}$.

For the purpose of discovering the structure of the stabilizer subgroup S_3 in S_2 we now fix $3\rightarrow 3$ and find the new current type matrix. As shown in Figure 4.0.5 the diagonal elements at this stage are all

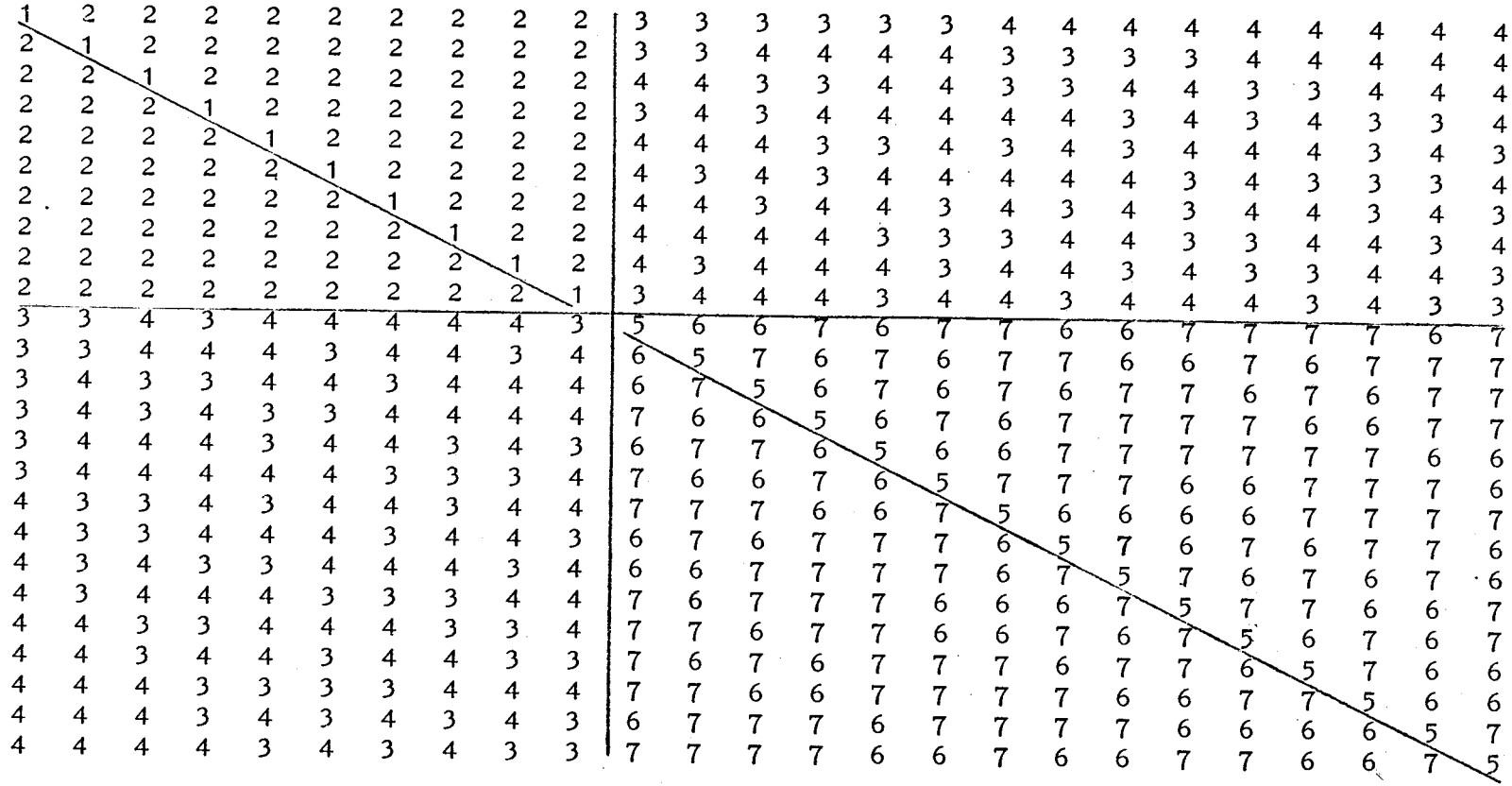


Figure 4.0.2 The type matrix of graph(ii) shown in Figure 3.0.1

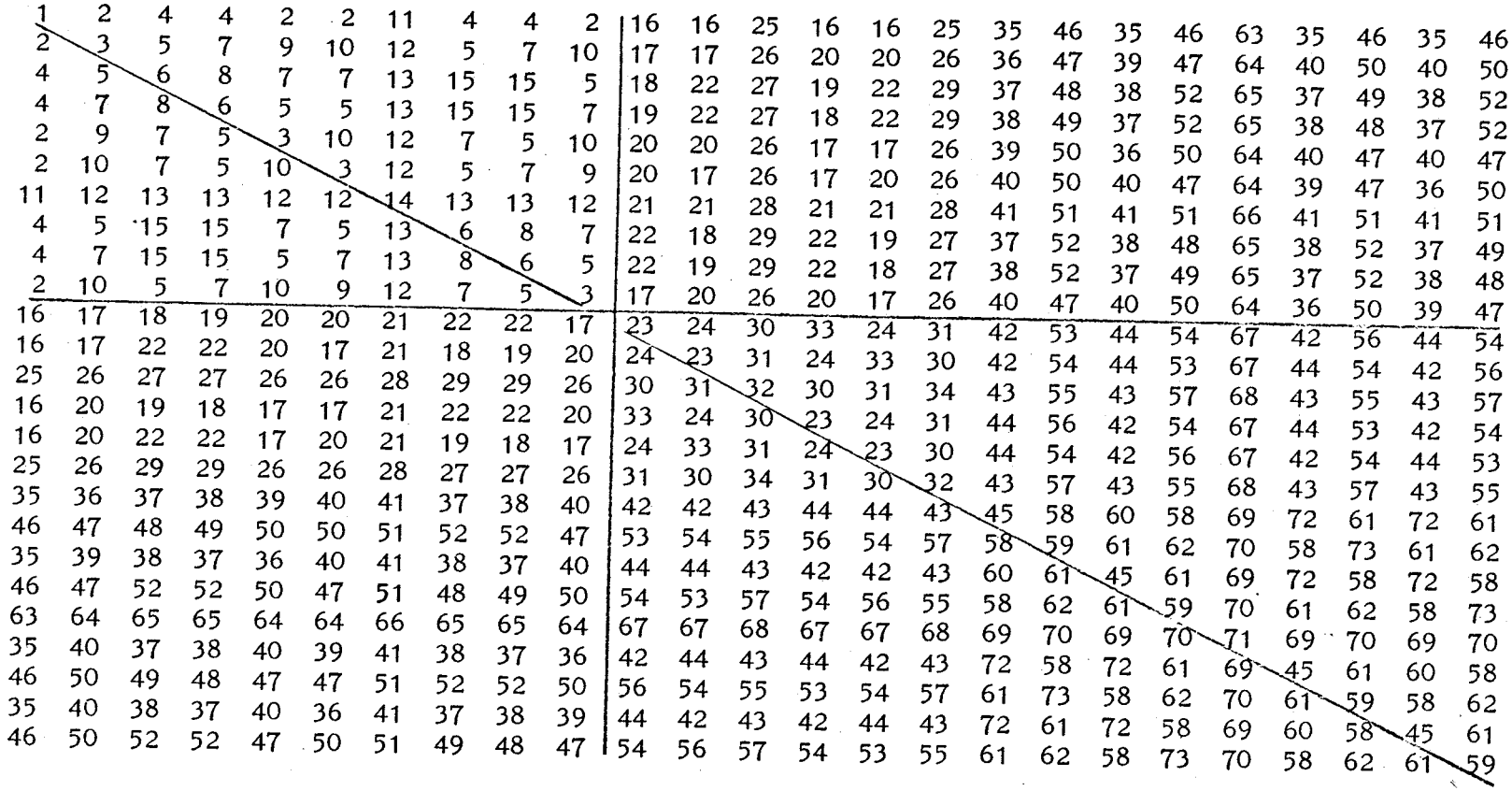


Figure 4.0.3 The Type matrix with node 1 fixed

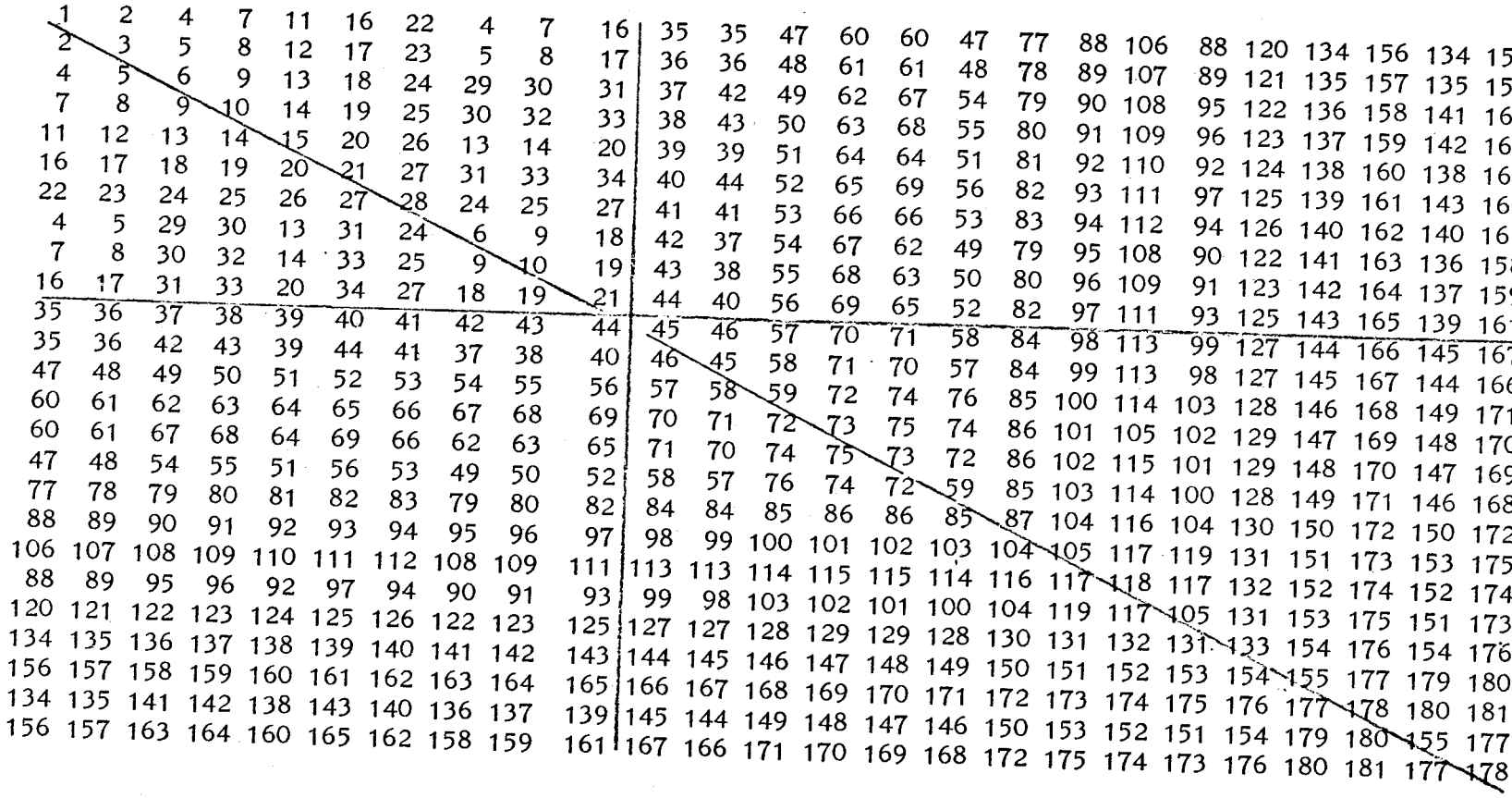


Figure 4.0.4 The type matrix with nodes 1 and 2 fixed

1	2	4	7	11	16	22	29	37	46	56	67	79	92	106	121	137	154	172	191	211	232	254	277	301
2	3	5	8	12	17	23	30	38	47	57	68	80	93	107	122	138	155	173	192	212	233	255	278	302
4	5	6	9	13	18	24	31	39	48	58	69	81	94	108	123	139	156	174	193	213	234	256	279	303
7	8	9	10	14	19	25	32	40	49	59	70	82	95	109	124	140	157	175	194	214	235	257	280	304
11	12	13	14	15	20	26	33	41	50	60	71	83	96	110	125	141	158	176	195	215	236	258	281	305
16	17	18	19	20	21	27	34	42	51	61	72	84	97	111	126	142	159	177	196	216	237	259	282	306
22	23	24	25	26	27	28	35	43	52	62	73	85	98	112	127	143	160	178	197	217	238	260	283	307
29	30	31	32	33	34	35	36	44	53	63	74	86	99	113	128	144	161	179	198	218	239	261	284	308
37	38	39	40	41	42	43	44	45	54	64	75	87	100	114	129	145	162	180	199	219	240	262	285	309
46	47	48	49	50	51	52	53	54	55	65	76	88	101	115	130	146	163	181	200	220	241	263	286	310
56	57	58	59	60	61	62	63	64	65	66	77	89	102	116	131	147	164	182	201	221	242	264	287	311
67	68	69	70	71	72	73	74	75	76	77	78	90	103	117	132	148	165	183	202	222	243	265	288	312
79	80	81	82	83	84	85	86	87	88	89	90	91	104	118	133	149	166	184	203	223	244	266	289	313
92	93	94	95	96	97	98	99	100	101	102	103	104	105	119	134	150	167	185	204	224	245	267	290	314
106	107	108	109	110	111	112	113	114	115	116	117	1 8	119	120	135	151	168	186	205	225	246	268	291	315
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	152	169	187	206	226	247	269	292	316
137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	170	188	207	227	248	270	293	317
154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	189	208	228	249	271	294	318
172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	209	229	250	272	295	319
191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	230	251	273	296	320
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	252	274	297	321
232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	275	298	322
254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	299	323
277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	324
301	302	303	304	305	306	307	308	309	310	311	312	313	314	135	316	317	318	319	320	321	322	323	324	325

Figure 4.0.5 The type matrix with nodes 1, 2 and 3 fixed

unique and we have reached the identity permutation I which is the smallest subgroup in the sequence. The following table is a summary of the various possibilities so far at each level:

Level	Group	Current Type Matrix	Fixed Nodes	Choices For Next Level
0	Γ_{G_1}	Fig. 4.0.2	none	$1 \rightarrow \{1,2,3,4,5,6,7,8,9,10\}$
1	S_1 in Γ_{G_1}	Fig. 4.0.3	1	$2 \rightarrow \{2,5,6,10\}$
2	S_2 in S_1	Fig. 4.0.4	1,2	$3 \rightarrow \{3,8\}$
3	S_3=I in S_2	Fig. 4.0.5	1,2,3	no choices

At this point the algorithm "backs-up" to find the generators of the other subgroups in the chain. For the stabilizer subgroup S_2 in S_1 there is one other possible choice: $3 \rightarrow 8$. This possibility is explored by making two copies of the current type matrix at level 2 (shown in Figure 4.0.4) and fixing the mapping $3 \rightarrow 8$ by assigning the integer value 182 to the third diagonal element of the first matrix and the eightth diagonal element of the second matrix. The resultant type matrix is shown in Figures 4.0.6a and 4.0.6b. Since this matrix has unique diagonal elements we have discovered the mapping: γ_1= (1) (2) (3 8) (4 9) (5) (6 10) (7) (11 12) (13 16) (14 15) (17) (18 20) (19) (21) (22 24) (23 25) which is indeed an automorphism. The algorithm double checks that the mapping is an automorphism via multiplication with the original adjacency matrix of the graph. At level 2 there are no other possibilities so S_2 in S_1 consists of two automorphisms: γ_1 and I.

Backing-up to level 1 we must find a minimum number of elements of S_1 that will generate the different cosets of S_2 in S_1. Two copies of the type matrix at level 1 (the one shown in Figure 4.0.3) are made

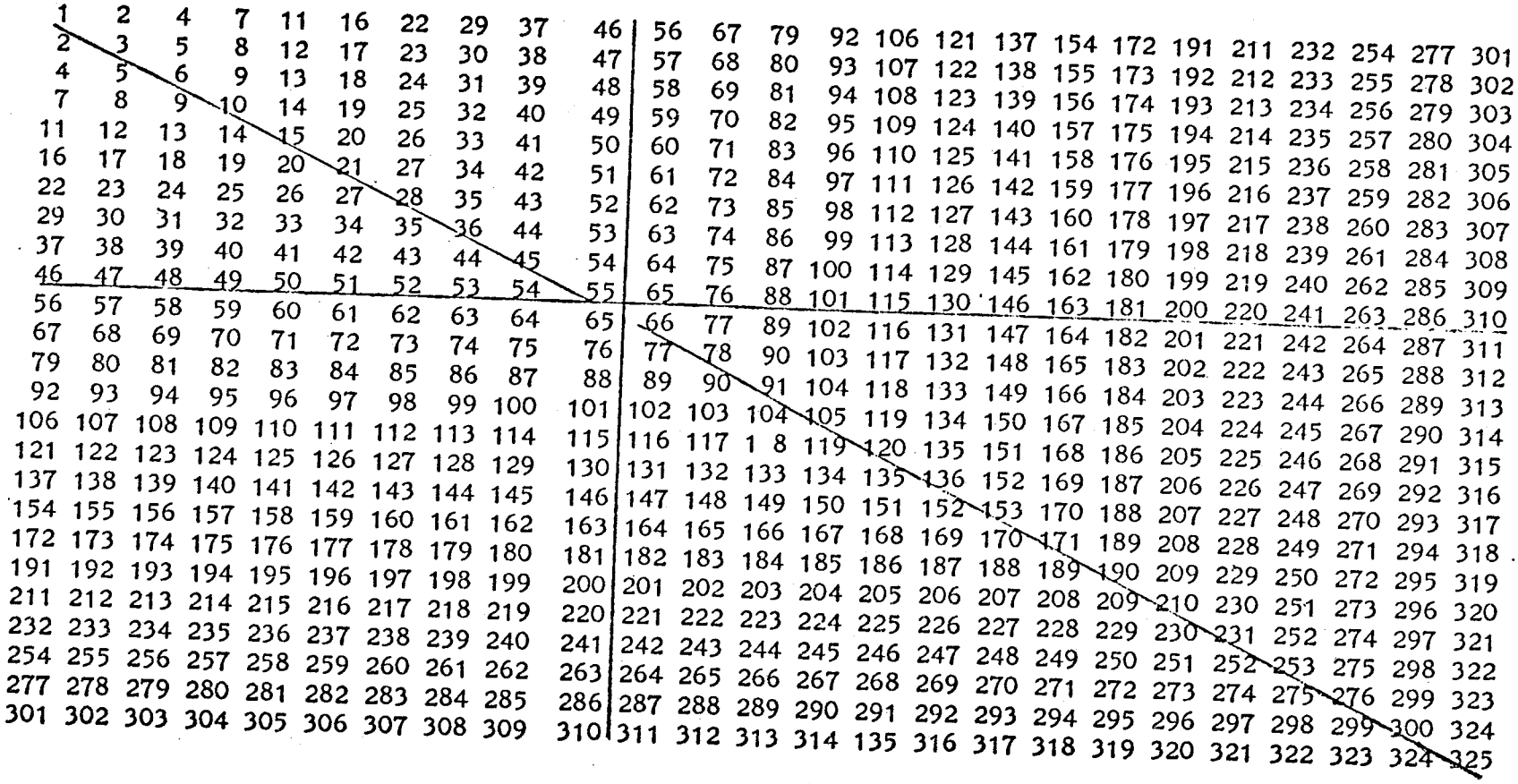


Figure 4.0.6a The first type matrix for fixing 1→1, 2→2, and 3→8.

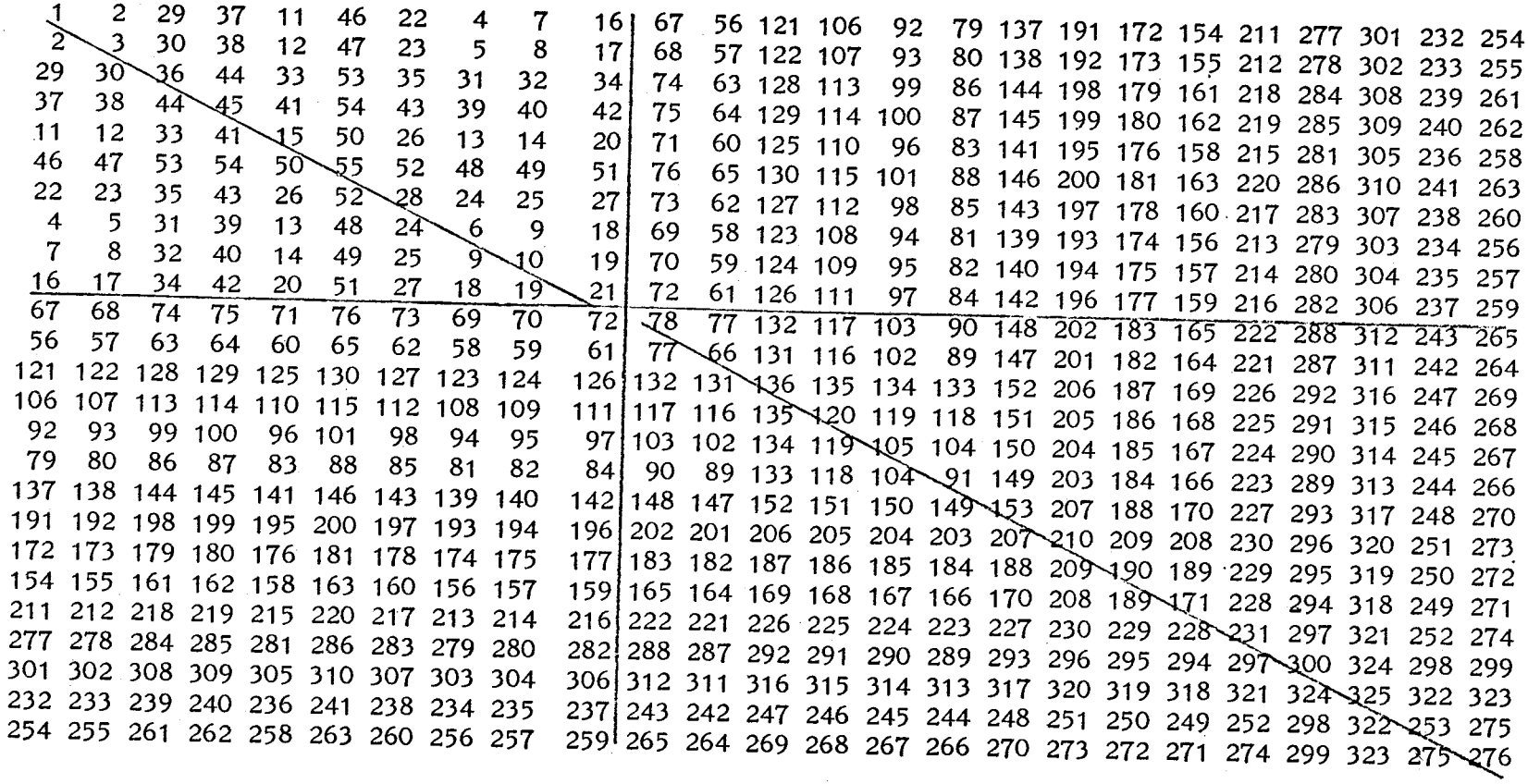


Figure 4.0.6b The second type matrix for fixing $1 \to 1$, $2 \to 2$, and $3 \to 8$.

and the mapping $2\rightarrow 5$ is fixed by setting the second diagonal element of the first matrix and the fifth diagonal element of the second matrix equal to the integer 74. The type matrices of these modified matrices are computed and shown in Figures 4.0.7a and 4.0.7b. At this stage there is still some choice. Ideally it would be desirable to make the $2\rightarrow 5\rightarrow \ldots$ orbit as large as possible so that we could include a maximum number of the possibilities for this level in one permutation. In general the algorithm will always make the choice to give a large orbit; however, in this instance there is no further choice for node 5 so we select $3\rightarrow 4$. This yields the type matrices shown in Figures 4.0.8a and 4.0.8b which provide the automorphism γ_2 = (1) (2 5) (3 4) (6 10) (7) (8 9) (11 14) (12 15) (13) (16) (17 19) (18 23) (20 25) (21) (22 24). So γ_2 will generate one of the cosets of S_2 in S_1.

The next choice at level 1 is $2\rightarrow 6$. The method proceeds in the same fashion and produces the automorphism γ_3 = (1) (2 6 5 10) (3 8 4 9) (7) (11 12 14 15) (13 16) (17 24 19 22) (18 20 23 25) (21) which will generate all the possibilities at level 1. Hence the index of S_2 in S_1 is 4 and so S_1 consists of 8 automorphisms.

Lastly at level 0 we must determine which of $\{2,3,4,5,6,7,8,9,10\}$ node 1 may map onto. Two copies of the type matrix at level 0 (shown in Figure 4.0.2) are formed and the mapping $1\rightarrow 2$ is fixed. In this case the typing process discovers that $1\rightarrow 2$ is not a possible mapping. Since $2\rightarrow\{2,6,5,10\}$ in γ_3 the algorithm deduces that 1 cannot map onto $\{2,5,6,10\}$ and so excludes these from the possibilities at level 0. The next possibility at level 0 is $1\rightarrow 3$ which after processing produces the automorphism γ_4 = (1 3 7 4) (2 5 6 10) (8 9) (11 14 18 23) (12 17 25 24) (13) (15 22 20 19) (16 21). Cross checking with the orbits

1	2	4	7	11	16	22	4	7	16	35	35	47	60	60	47	77	88	106	88	120	134	156	134	156
2	3	5	8	12	17	23	5	8	17	36	36	48	61	61	48	78	89	107	89	121	135	157	135	157
4	5	6	9	13	18	24	29	30	31	37	42	49	62	67	54	79	90	108	95	122	136	158	141	163
7	8	9	10	14	19	25	30	32	33	38	43	50	63	68	55	80	91	109	96	123	137	159	142	164
11	12	13	14	15	20	26	13	14	20	39	39	51	64	64	51	81	92	110	92	124	138	160	138	160
16	17	18	19	20	21	27	31	33	34	40	44	52	65	69	56	82	93	111	97	125	139	161	143	165
22	23	24	25	26	27	28	24	25	27	41	41	53	66	66	53	83	94	112	94	126	140	162	140	162
4	5	29	30	13	31	24	6	9	18	42	37	54	67	62	49	79	95	108	90	122	141	163	136	158
7	8	30	32	14	33	25	9	10	19	43	38	55	68	63	50	80	96	109	91	123	142	164	137	159
16	17	31	33	20	34	27	18	19	21	44	40	56	69	65	52	82	97	111	93	125	143	165	139	161
35	36	37	38	39	40	41	42	43	44	45	46	57	70	71	58	84	98	113	99	127	144	166	145	167
35	36	42	43	39	44	41	37	38	40	46	45	58	71	70	57	84	99	113	98	127	145	167	144	166
47	48	49	50	51	52	53	54	55	56	57	58	59	72	74	76	85	100	114	103	128	146	168	149	171
60	61	62	63	64	65	66	67	68	69	70	71	72	73	75	74	86	101	115	102	129	147	169	148	170
60	61	67	68	64	69	66	62	63	65	71	70	74	75	73	72	86	102	115	101	129	148	170	147	169
47	48	54	55	51	56	53	49	50	52	58	57	76	74	72	59	85	103	114	100	128	149	171	146	168
77	78	79	80	81	82	83	79	80	82	84	84	85	86	86	85	87	104	116	104	130	150	172	150	172
88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	117	119	131	151	173	153	175
106	107	108	109	110	111	112	108	109	111	113	113	114	115	115	114	116	117	118	117	132	152	174	152	174
88	89	95	96	92	97	94	90	91	93	99	98	103	102	101	100	104	119	117	105	131	153	175	151	173
120	121	122	123	124	125	126	122	123	125	127	127	128	129	129	128	130	131	132	131	133	154	176	154	176
134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	177	179	180
156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	180	181
134	135	141	142	138	143	140	136	137	139	145	144	149	148	147	146	150	153	152	151	154	179	180	155	177
156	157	163	164	160	165	162	158	159	161	167	166	171	170	169	168	172	175	174	173	176	180	181	177	178

Figure 4.0.7a The first type matrix for 1→1 and 2→5.

1	11	7	4	2	16	22	7	4	16	60	60	47	35	35	47	106	156	77	156	120	134	88	134	88
11	15	14	13	12	20	26	14	13	20	64	64	51	39	39	51	110	160	81	160	124	138	92	138	92
7	14	10	9	8	33	25	32	30	19	63	68	50	38	43	55	109	159	80	164	123	142	91	137	96
4	13	9	6	5	31	24	30	29	18	62	67	49	37	42	54	108	158	79	163	122	141	90	136	95
2	12	8	5	3	17	23	8	5	17	61	61	48	36	36	48	107	157	78	157	121	135	89	135	89
16	20	33	31	17	21	27	19	18	34	69	65	56	44	40	52	111	165	82	161	125	139	97	143	93
22	26	25	24	23	27	28	25	24	27	66	66	53	41	41	53	112	162	83	162	126	140	94	140	94
7	14	32	30	8	19	25	10	9	33	68	63	55	43	38	50	109	164	80	159	123	137	96	142	91
4	13	30	29	5	18	24	9	6	31	67	62	54	42	37	49	108	163	79	158	122	136	95	141	90
16	20	19	18	17	34	27	33	31	21	65	69	52	40	44	56	111	161	82	165	125	143	93	139	97
60	64	63	62	61	69	66	68	67	65	73	75	72	70	71	74	115	169	86	170	129	148	101	147	102
60	64	68	67	61	65	66	63	62	69	75	73	74	71	70	72	115	170	86	169	129	147	102	148	101
47	51	50	49	48	56	53	55	54	52	72	74	59	57	58	76	114	168	85	171	128	149	100	146	103
35	39	38	37	36	44	41	43	42	40	70	71	57	45	46	58	113	166	84	167	127	145	98	144	99
35	39	43	42	36	40	41	38	37	44	71	70	58	46	45	57	113	167	84	166	127	144	99	145	98
47	51	55	54	48	52	53	50	49	56	74	72	76	58	57	59	114	171	85	168	128	146	103	149	100
106	110	109	108	107	111	112	109	108	111	115	115	114	113	113	114	118	174	116	174	132	152	117	152	117
156	160	159	158	157	165	162	164	163	161	169	170	168	166	167	171	174	178	172	181	176	180	173	177	175
77	81	80	79	78	82	83	80	79	82	86	86	85	84	84	85	116	172	87	172	130	150	104	150	104
156	160	164	163	157	161	162	159	158	165	170	169	171	167	166	168	174	181	172	178	176	177	175	180	173
120	124	123	122	121	125	126	123	122	125	129	129	128	127	127	128	132	176	130	176	133	154	131	154	131
134	138	142	141	135	139	140	137	136	143	148	147	149	145	144	146	152	180	150	177	154	155	153	179	151
88	92	91	90	89	97	94	96	95	93	101	102	100	98	99	103	117	173	104	175	131	153	105	151	119
134	138	137	136	135	143	140	142	141	139	147	148	146	144	145	149	152	177	150	180	154	179	151	155	153
88	92	96	95	89	93	94	91	90	97	102	101	103	99	98	100	117	175	104	173	131	151	119	153	105

Figure 4.0.7b The second type matrix for 1→1 and 2→5.

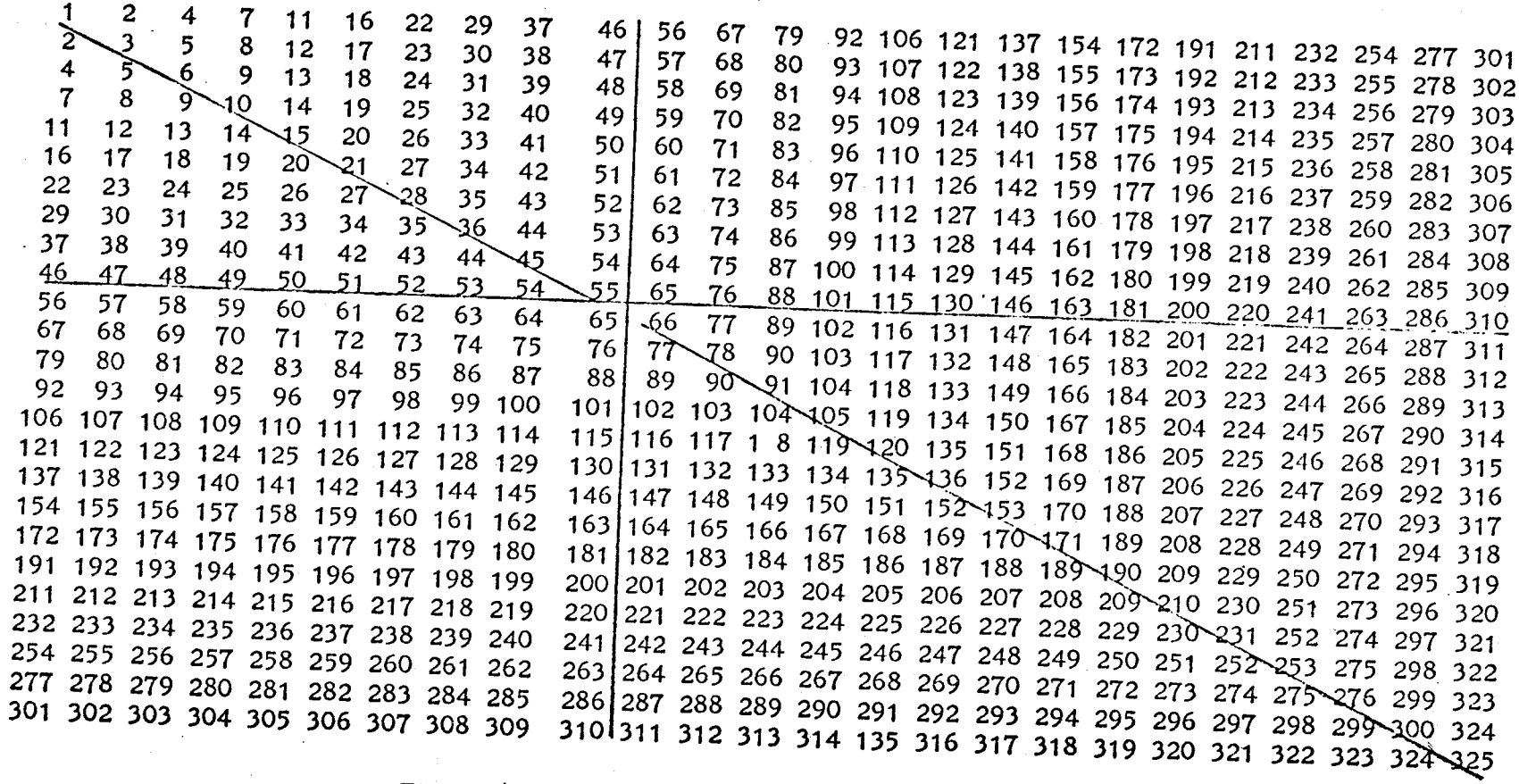


Figure 4.0.8a The first type matrix fixing 1→1, 2→5, and 3→4.

1	11	7	4	2	46	22	37	29	16	92	106	79	56	67	121	172	254	137	301	211	277	154	232	191
11	15	14	13	12	50	26	41	33	20	96	110	83	60	71	125	176	258	141	305	215	281	158	236	195
7	14	10	9	8	49	25	40	32	19	95	109	82	59	70	124	175	257	140	304	214	280	157	235	194
4	13	9	6	5	48	24	39	31	18	94	108	81	58	69	123	174	256	139	303	213	279	156	234	193
2	12	8	5	3	47	23	38	30	17	93	107	80	57	68	122	173	255	138	302	212	278	155	233	192
46	50	49	48	47	55	52	54	53	51	101	115	88	65	76	130	181	263	146	310	220	286	163	241	200
22	26	25	24	23	52	28	43	35	27	98	112	85	62	73	127	178	260	143	307	217	283	160	238	197
37	41	40	39	38	54	43	45	44	42	100	114	87	64	75	129	180	262	145	309	219	285	162	240	199
29	33	32	31	30	53	35	44	36	34	99	113	86	63	74	128	179	261	144	308	218	284	161	239	198
16	20	19	18	17	51	27	42	34	21	97	111	84	61	72	126	177	259	142	306	216	282	159	237	196
92	96	95	94	93	101	98	100	99	97	105	119	104	102	103	134	185	267	150	314	224	290	167	245	204
106	110	109	108	107	115	112	114	113	111	119	120	118	116	117	135	186	268	151	215	225	291	168	246	205
79	83	82	81	80	88	85	87	86	84	104	118	91	89	90	133	184	266	149	313	223	289	166	244	203
56	60	59	58	57	65	62	64	63	61	102	116	89	66	77	131	182	264	147	311	221	287	164	242	201
67	71	70	69	68	76	73	75	74	72	103	117	90	77	78	132	183	265	148	312	222	288	165	243	202
121	125	124	123	122	130	127	129	128	126	134	135	133	131	132	136	187	269	152	316	226	292	169	247	206
172	176	175	174	173	181	178	180	179	177	185	186	184	182	183	187	190	272	188	319	229	295	189	250	209
254	258	257	256	255	263	260	262	261	259	267	268	266	264	265	269	272	276	270	323	274	299	271	275	273
137	141	140	139	138	146	143	145	144	142	150	151	149	147	148	152	188	270	153	317	227	293	170	248	207
301	305	304	303	302	310	307	309	308	306	314	315	313	311	312	316	319	323	317	325	321	324	318	322	320
211	215	214	213	212	220	217	219	218	216	224	225	223	221	222	226	229	274	227	321	231	297	228	252	230
277	281	280	279	278	286	283	285	284	282	290	291	289	287	288	292	295	299	293	324	297	300	294	298	296
154	158	157	156	155	163	160	162	161	159	167	168	166	164	165	169	189	271	170	318	228	294	171	249	208
232	236	235	234	233	241	238	240	239	237	245	246	244	242	243	247	250	275	248	322	252	298	249	253	251
191	195	194	193	192	200	197	199	198	196	204	205	203	201	202	206	209	273	207	320	230	296	208	251	210

Figure 4.0.8b The second type matrix fixing 1→1, 2→5, and 3→4.

of γ_2 and γ_3 reveals that all the possibilities for level 0 are included. Since $\{1,3,7,4,8,9\}$ are the possibilities for node 1, the index of S_1 in Γ_{G_1} is 6, and the order of the whole group is $6 \times 8 = 48$.

The same process applied to graph(i) of Figure 3.0.1 finds the following four generators for the whole automorphism group of order 720:

γ_1 = (1) (2) (3) (4 6) (5 7) (8) (9 10) (11 12) (13 14) (15 16) (17 18) (19 20) (21 22) (23) (24) (25)

γ_2 = (1) (2) (3 5 8 7) (4 6 10 9) (11 12) (13 14 15 16) (17 19 20 18) (21 23 22 25) (24)

γ_3 = (1) (2 3 8) (4 5 9) (6 7 10) (11 14 16) (15 12 13) (17 21 19) (18 22 20) (23 25 24)

γ_4 = (1 2 3 4 5) (6 10 9 7 8) (11 17 13 19 14) (12 18 21 23 15) (16 20 22 24 25).

4.1 The Automorphism Group Algorithm

In this section we give a more explicit statement of the algorithm to determine the generators of the automorphism group of a graph G_1. Initially the method establishes a depth first level of possibilities for the nested chain of stabilizer subgroups right down to the identity permutation by fixing one node at a time and recording the possibilities at each level in a stack. This process continues until the identity mapping is found and there are no other alternatives. At this point the algorithm successively backs-up the tree of possibilities one level at a time to find the generators of the automorphism group.

Algorithm 4.1.1 The Automorphism Finder

Step 1: Initialize the number of generators found so far to zero: NUMPERM:=0. Initialize the stack pointer β to 0. Let STACKMATRIX(0) contain the type matrix $T_\delta(A_1^*)$ of G_1. Initialize the automorphism group counter GROUPORDER:=1. Let HIGHINTEGER(0) contain the largest integer of the type matrix contained in STACKMATRIX(0).

Step 2: (Setting up possible mappings at level β).
Let STACKLIST(β) contain an ascending list of integers $\{a_{\beta,1}, a_{\beta,2}, \ldots\}$ such that these integers are the position numbers of the smallest group of repeating diagonal elements (within the set we are considering as a transitive subsystem) in the type matrix contained in STACKMATRIX(β). Let STACKNUM(β) contain the order of the above set $\{a_{\beta,1}, a_{\beta,2}, \ldots\}$.
Initialize the count of rejected possibilities to zero: REJCOUNT(β):=0.

Step 3: (Fix a mapping for the next level.)
Increase the stack pointer: $\beta := \beta+1$.
Let WORKMATRIX = STACKMATRIX($\beta-1$) with the modification that the $a^{th}_{\beta-1,1}$ diagonal element is given the value HIGHINTEGER($\beta-1$)+1.

Let STACKMATRIX(β) be the type matrix of WORKMATRIX. Let HIGHINTEGER(β) be the largest integer in STACKMATRIX(β).

Step 4: If there are some repeating diagonal elements in STACKMATRIX(β) then step 2.

Step 5: (The identity mapping is established so begin filtering back up the chain of nested subgroups to find the generators of the cosets at each level.)
Decrease the stack pointer $\beta = \beta - 1$.

Step 6: (Try to find generators of all the possibilities at level β).
Let $\gamma = a_{\beta,2}$.

Step 7: (Mapping $a_{\beta,1} \rightarrow \gamma$).
Let WORKMATRIX AND WORKMATRIX2 equal STACKMATRIX(β).
Let ϵ = HIGHINTEGER(β).
Set the $a_{\beta,1}^{th}$ diagonal element of WORKMATRIX equal to $\epsilon+1$.
Set the γ^{th} diagonal element of WORKMATRIX2 equal to $\epsilon+1$.

Step 8: Use the isomorphism algorithm of Chapter 3 to find a mapping $a_{\beta,1} \rightarrow \gamma$ (recall that the method will try to make the orbit of the permutation containing $a_{\beta,1}$ and γ as large as possible).
If a mapping P is found then step 10.

Step 9: Otherwise increase the count of rejected mappings for this level. REJCOUNT(β) = REJCOUNT(β) + 1 and REJLIST(REJCOUNT(β)) = γ.
Decrease the count of possibilities at this level by one: STACKNUM(β) = STACKNUM(β) - 1. Search through the list of automorphisms already found at lower levels and regard any elements in the same orbit as γ as impossible for this level. Where necessary remove these from list STACKLIST decrementing STACKNUM and add them to REJLIST incrementing the counter REJCOUNT accordingly. Set γ = the next element in the list STACKLIST(β) and go to step 7. If STACKLIST(β) is exhausted then step 13.

Step 10: Increase the count of the number of automorphisms found so far, NUMPERM:=NUMPERM+1. Save the automorphism P, PERMUTATIONS(NUMPERM) = P.

Step 11: Regard any other nodes the orbit with $a_{\beta,1}$ and γ or any nodes in an orbit with γ from an automorphism at a lower level as having been found within the list STACKLIST(β). Similarily regard any of the impossible choices for these nodes as impossible for $a_{\beta,1}$ and add it to its list REJLIST(β) while decrementing REJCOUNT(β) and STACKNUM(β).

Step 12: Let γ = the next element in the list STACKLIST(β) and go to step 7 if the list is not exhausted..

Step 13: (The possibilities at level β are exhausted so increase the group order and backup another level.)
GROUPORDER:=GROUPORDER*STACKNUM(β).
$\beta := \beta - 1$.
IF $\beta \geq 0$ THEN step 6 ELSE TERMINATE.

As seen in section 4.0 the above algorithm applies a powerful algebraic method to find a small set of generators for the automorphism group. However, we cannot claim as McKay [1976] does, that it will determine an absolutely minimal number of generators. The number of generators that are discovered will depend largely on the structure of the nested chain of stabilizer subgroups. The number of generators found will determine the computation time and so it is somewhat meaningless to try and predict what this time will be. The set of eight 25-point strongly regular graphs are given in Appendix II along with the generators of their automorphism groups as determined by this algorithm.

It is worthwhile noting here that the basic strategy of the above algorithm may be varied. For example instead of beginning with the smallest node set of candidates for a transitive subgraph we could select the largest such subset. Although varying the search strategy may have a significant effect on any given graph it is believed that over the set of all graphs the results would be approximately similar.

CHAPTER 5

Evaluation

5.0 Overview

The evaluation of a new algorithm includes a prediction of the order of the computation, a statement clarifying which sub-class(es) of all the possible inputs the algorithm will process successfully, and where possible a comparison with existing algorithms. The discussion contained in this chapter is aimed at providing answers pertaining to all of the above considerations with respect to the isomorphism algorithms proposed in this thesis.

As mentioned in Chapter 1, we maintain that a testing of the algorithms on randomly generated graphs will shed little light on the algebraic power of the method to detect non-isomorphism of cospectral graphs. For the probability that two randomly generated graphs on n points are cospectral is very small. Recall that Harary et al. [1971] remark that it is tempting to conjecture that this probability is increasingly smaller as n increases. Clearly when our algorithm is directed at cospectral graphs, measuring the non-isomorphism detection capability on randomly generated graphs will establish very little. Instead we have chosen to observe the performance of the algorithm in detecting non-isomorphism on some known "difficult" graphs. These include graphs of designs and strongly regular graphs communicated by D. Corneil and R. Mathon. In particular, the behaviour of the algorithm on a set of 25-point non-isomorphic strongly regular graphs serves as a valuable comparison with Druffel's algorithm on such a set of graphs (see section 1.5.4).

The isomorphism detection capacity of the algorithm is much easier to measure as any graph and a permutation of that same graph will serve

as a non-trivial example. However in order to predict the timing as n increases we have chosen sets of examples that have consistent properties as n increases. In one set for example, the timings for random permutations of the complete graphs are studied for n equal to 10, 12, 16, 20, 30, and 40.

The body of this chapter is divided into three main sections. The first concerns detection of non-isomorphism without the use of a type matrix, the second deals with the backtracking algorithm for either isomorphism or non-isomorphism based on the type matrix computation, and the third section pertains to the algorithm for determining the generators of the automorphism group of a graph.

5.1 Non-Isomorphism Detection Without the Type Matrix

In this section we consider the non-backtracking algorithms of Chapter 2 that are capable of detecting the non-isomorphism of cospectral graphs without resorting to the type matrix computation of Chapter 3. It is difficult to predict exactly which general class of non-isomorphic cospectral graphs are successfully handled by these algorithms; however some examples which fall into this class are given in the Figures of Chapter 2. It would appear that trees, all non-regular graphs that are not graphs of designs, and some regular graphs belong to this class. Certainly it is our experience that strongly regular graphs and graphs of designs do not fall into this category. It might be reasonable to conjecture that all cospectral graphs that are either strongly regular or have a strongly regular subgraph, and graphs of designs are indistinguishable via these algorithms. For these more "difficult" examples the backtracking Algorithm 3.3.1 will either find an isomorphism or disprove isomorphism. Therefore if one groups the collection of algorithms of this thesis together then it may be said that they will always determine

a correct solution for any input pair of cospectral graphs provided they are connected and without loops or multiple edges. The fact that a correct solution is guaranteed is an important positive attribute of this collection of algorithms.

Recall that the non-isomorphism detection algorithms of Chapter 2 first process the eigenvectors of the simple eigenvalues of the given graphs. If this fails to determine a "difference" between the graphs then Algorithm 2.3.1 is employed to process the eigenvectors of the multiple eigenvalues. In sections 2.2 and 2.3 it was established that the computation is of order $n^3 + n^2 + n^2\log n + n\log n + n$ for the simple eigenvalue case and of order $n^3\log n + n^3 + n^2\log n + n^2 + n\log n$ for the multiple eigenvalue case. It is evident that for some non-isomorphic cospectral graphs (including ones with highly multiple eigenvalues) the method is able to compute a correct solution in $n^3\log n$ time at worst.

5.2 The Backtracking Isomorphism Algorithm

In this section the backtracking algorithm is observed in its capacity to detect:

(i) isomorphism with no backtracking

(ii) non-isomorphism with backtracking

(iii) isomorphism with backtracking.

In order to confirm the theoretical analysis of the computation we present a summary of actual computer timings for sets of selected examples for cases (i) and (ii) above. It is argued that the timings for case (iii) will be dependent on case (ii).

Before discussing the complete algorithm it is necessary to establish the order of the computation in Algorithm 3.1 for determining the type matrices $T_{\mathcal{S}}(A_1^*)$ and $T_{\mathcal{S}}(A_2^*)$ of two given cospectral matrices A_1 and A_2. This process involves each one of the following major steps:

(1) Let matrices T_1 and T_2 equal input matrices A_1 and A_2 respectively. Computing the eigenvalues and eigenvectors of T_1 and T_2 requires $O(n^3)$ time.

(2) Sorting the eigenvalues by a heapsort requires O(nlogn) comparisons.

(3) For each one of the k distinct eigenvalues, the matrices H_i and H_i' for i=1,2,...,k in the matrix decomposition of T_1 and T_2, must be computed. Since each H_i matrix is nxn and symmetric, then its formation requires $m_i x(n^2 + n)/2$ multiplications. Since $\sum_{i=1}^{k} m_i = n$ then the total multiplications for forming all the H_i matrices is $(n^3 + n^2)/2$.

(4) For each one of the k pairs of matrices H_i and H_i' there is a refinement of T_1 and T_2 based on the integers appearing in T_1 and T_2 together with the real numbers appearing in the corresponding positions in H_i and H_i' respectively. Assume that at the beginning of the j^{th} refinement the matrices T_1 and T_2 both contain r different integers (for if they both do not contain the same number of integers then the typing algorithm will recognize them as being different and quit). Further suppose that H_j and H_j' contain s different real numbers. For each one of the r different integers in T_1 and T_2, the algorithm will setup a binary tree in which to store the real numbers from H_j and H_j' that appear in the corresponding positions. Let $\beta_1, \beta_2, \ldots, \beta_r$ denote these binary trees each containing $\delta_1, \delta_2, \ldots, \delta_r$ members respectively. It is obvious that $\sum_{p=1}^{r} \delta_p$ is bounded by rxs, which in turn is bounded by $(n^2 + n)/2$ since the matrices are symmetric. Hence to refine each one of the $(n^2 + n)/2$ positions in the lower triangles of T_1 and T_2, the algorithm must search for or insert a real number in a tree of maximum length $(n^2 + n)/2$. So the j^{th} refinement stage requires $O(n^2 + n)^2 \approx O(n^4 + n^3 + n^2)$ comparisons at worst.

Since k (the number of distinct eigenvalues) has an upper bound of n, the total number of comparisons over all the refinement stages is of order $n^5 + n^4 + n^2$. It is worthwhile noting here that for graphs where the multiplicty of the eigenvalues is high (ie. when k is small) the above computation remains of order n^4. The strongly regular graphs and graphs of designs that we have processed tend to have eigenvalues of high multiplicity and so for these cases the type matrix computation is expected to tend toward n^4 time rather than n^5 time. Similarily the complete graph K_n on n points has eigenvalues n-1 of multiplicity 1 and eigenvalue -1 of multiplicity n-1. So in this case one would expect the type matrix computation to be of order n^4.

In practice when the program types the two input matrices A_1 and A_2, it continues re-typing the type matrices until the total number of different integers in the type matrices remains constant. It was observed during the computer runs that most often the typing was done only twice. Sometimes it was repeated three times and the following circumstance is the condition under which this occurred: if upon calculating a type matrix for the first time there exists two columns i and j that have the same element on the main diagonal but a differing set of numbers in the rest of the column then this situation seems to cause an increase in the total number of different integers upon re-typing. The apparent effect of the re-typing is to distinguish between columns i and j by placing different integers in the i^{th} and j^{th} positions along the main diagonal. Intuitively speaking this re-typing of the types cannot occur very many times before either a complete mapping is determined or the type matrices are discovered to be different.

At this point we will consider the order of the computation for Algorithm 3.3.1 to detect an isomorphism when there is no backtracking.

In order to provide a meaningful timings study of this algorithm, three different sets of graphs have been selected, each containing one graph on 10, 12, 16, 20, 30, and 40 points respectively. The examples are such that each set is homogeneous to a certain degree with respect to graphical properties. The first set contains all the complete graphs over the above range of points. There is certainly no better example of a graph which maintains the same graphical properties as n increases. Other than for a timing study, the complete graph is not an interesting isomorphism example because it is easily recognizable by either its degree sequence or its spectrum.

In the case of no backtracking to detect isomorphism, Algorithm 3.3.1 simply selects (by sorting of the main diagonal elements from the successive type matrices) a pair of nodes to establish a partial mapping, re-computes the type matrix, and continues fixing further pairs of nodes until all the diagonal elements are unique when there is no further choice and so the isomorphic mapping is established between the two matrices. To verify the isomorphism, the algorithm does a check by multiplying the original matrices by the representative permutation matrix. The process of selecting a new pair of nodes to fix at each stage is of order nlogn since it merely involves sorting of the n main diagonal elements. It is easily seen that the type matrix computation is the dominating factor in the timing. Consequently the number of mappings that have to be "fixed" before an isomorphism is found determines the order of the computation (since each mapping that is fixed invokes a computation of the type matrix). The number of times that this occurs depends upon the structure of the graph. If the graphs are rigid then it will occur only once, and if the graphs are complete for example, it is necessary to fix n-1 nodes before

the isomorphism is discovered. Certainly n-1 is an upper limit on the number of forced mappings.

Let us now consider the timings for the set of complete graphs. One might be tempted to classify these graphs as an easy example because they are so highly transitive; however, in a sense these are "worst case" graphs in that they require the maximum number of forced mappings to find an isomorphism. Since the multiplicity of the eigenvalues for a complete graph is very high, the computation to force n-1 nodes is expected to be of order $n*n^4 = n^5$. The results are presented in the form of a table with the actual mean computing times occurring in the left hand column and the extrapolated timings based on the actual mean time of each order of n is contained in the other six columns. The mean times were measured by testing up to twenty cases of random permutations of the complete graph on the given number of points. For n=40 only one timing was done.

n	actual mean time (sec.)	Predicted n^4 time based on actual mean time for n = 10	12	16	20	30	40
10	2.37		2.04	1.64	1.50	1.36	1.35
12	4.23	4.91		3.41	3.10	2.81	2.81
16	10.76	15.53	13.37		9.81	8.89	8.88
20	23.94	37.92	32.64	26.27		21.71	21.68
30	109.93	191.97	165.23	132.99	121.20		109.69
40	346.68	606.72	522.22	420.31	383.04	347.43	

Table 5.2.1 N^4 Prediction Table Based on Isomorphism Finder for Complete Graphs.

Clearly the n^4 predictions based on the actual mean time for n= 10 provide a good upper bound relative to the real mean times. As a consequence the extrapolation backwards of the actual mean times for increasing n causes a slight underestimate of the times for lower order n.

Since the estimate based on n = 10 seemed high as n increases the following table of estimates based on $n^3 \log n$ time was computed. Quite obviously it badly underestimates for n=40 but fits very well for low order n. Since our purpose is to establish an upper bound, the n^4 prediction table is more suitable. Clearly in this case n^5 is too high an upper bound.

n	actual mean time (sec.)	Predicted $n^3 \log n$ time based on actual mean time for n = 10	12	16	20	30	40
10	2.37		2.27	2.18	2.30	2.76	3.38
12	4.23	4.42		4.07	4.29	5.14	6.31
16	10.76	11.69	11.18		11.34	13.59	16.68
20	23.94	24.66	23.61	22.71		28.69	35.19
30	109.93	94.52	90.47	87.00	91.73		134.85
40	346.68	243.00	232.57	223.69	235.83	282.61	

Table 5.2.2 $N^3 \log N$ Prediction Table Based on Isomorphism Finder for Complete Graphs.

The second set of examples that were studied is a set of trees on 10, 12, 16, 20, 30, and 40 points respectively. Here again the prediction table based on a conjectured n^4 time provides a good upper bound compared with the real times as n increases. Table 5.2.3 contains the $n^3 \log n$ predictions for these trees and Table 5.2.4 contains the n^4 predictions.

The third set of examples is a study of the timings for detecting isomorphism of random permutations of Hamiltonian cubic graphs on the given range of points. Table 5.2.5 gives the extrapolated timings based on n^3. Evidently the n^3 bound grows too fast with increasing n for these cases, so the predictions were re-computed based on $n^2 \log n$ (shown

n	actual mean time (sec.)	Predicted $n^3 \log n$ time based on actual mean time for n = 10	12	16	20	30	40
10	1.12		1.21	1.66	1.49	0.93	1.32
12	2.26	2.09		3.09	2.78	1.73	2.47
16	8.17	5.52	5.98		7.33	4.57	6.53
20	15.46	11.67	12.61	17.24		9.67	13.78
30	36.97	44.67	48.33	66.07	59.24		52.81
40	135.77	114.84	124.26	169.84	152.30	95.04	

Table 5.2.3 $N^3 \log N$ Prediction Table Based on Isomorphism Finder for the set of Trees.

n	actual mean time (sec.)	Predicted n^4 time based on actual mean time for n = 10	12	16	20	30	40
10	1.12		1.09	1.25	0.97	0.46	0.53
12	2.26	2.32		2.59	2.00	0.95	1.10
16	8.17	7.34	7.14		6.33	2.99	3.48
20	15.46	17.92	17.44	19.95		7.30	8.49
30	36.97	90.72	88.28	100.98	78.27		42.96
40	135.77	286.72	279.01	319.14	247.36	116.84	

Table 5.2.4 N^4 Prediction Table Based on Isomorphism Finder for the set of Trees.

in Table 5.2.6). These timings match the real times more closely but still seem to grow too fast for increasing n. This would suggest that indeed the bound is something less than n^3 for these cases.

n	actual mean time (sec.)	Predicted n^3 time based on actual mean time for n = 10	12	16	20	30	40
10	1.57		1.35	1.20	1.58	0.51	0.28
12	2.34	2.71		2.07	2.72	0.88	0.48
16	4.90	6.43	5.55		6.45	2.09	1.13
20	12.60	12.56	10.83	9.57		4.08	2.21
30	13.77	42.39	36.56	32.30	42.53		7.47
40	17.70	100.48	86.67	76.56	100.80	32.64	

Table 5.2.5 N^3 Prediction Table Based on Isomorphism Finder for Hamiltonian Cubic Graphs

n	actual mean time (sec.)	Predicted $n^2 \log n$ time based on actual mean time for n = 10	12	16	20	30	40
10	1.57		1.51	1.59	2.42	1.04	0.70
12	2.34	2.44		2.47	3.76	1.61	1.07
16	4.90	4.84	4.64		7.46	3.19	2.13
20	12.60	8.17	7.84	8.27		5.39	3.59
30	13.77	20.87	20.02	21.13	32.19		9.18
40	17.70	40.24	38.60	40.75	62.06	26.55	

Table 5.2.6 $N^2 \log N$ Prediction Table Based on Isomorphism Finder for Hamiltonian Cubic Graphs

At this stage we wish to consider the backtracking algorithm in its capacity to detect non-isomorphism using backtracking. The two graphs of designs appearing in Figure 3.0.1 cause the algorithm to backtrack in its proof that these two graphs are not isomorphic. Similarily the sets of eight 25-point strongly regular graphs, ten 26-point strongly regular graphs, two 27-point strongly regular graphs, and two 40-point 3-strongly regular graphs all cause backtracking in the algorithm to disprove isomorphism. However all of these examples backtrack only n times at the top level. By this we mean that the algorithm successively tries to force the mappings $1\rightarrow 1$, $1\rightarrow 2$, ..., $1\rightarrow n$ and each time the typing algorithm immediately rejects these partial mappings as impossible. If we conjecture that any pair of non-isomorphic cospectral graphs will cause at most n backtracks at the top level to disprove isomorphism then the computation time would be as follows. Each one of the top level forced mappings would invoke a type matrix computation and so the total order of the computation for performing n backtracks at the top level would be $n*(n^5 + n^4 + n^2) = n^6 + n^5 + n^3$ which for strongly regular graphs is an acceptable computation time. If however an example that would cause backtracking at a deeper level of the tree were found then this computation time would be more.

For the eight strongly regular graphs on 25-points the algorithm to detect non-isomorphism was tried on all 28 different pairs of different ones from this set. Each time the algorithm proved non-isomorphism using 25 backtracks in a mean time of 31.71 seconds over the 28 examples. Compared with a minimum of 1825 backtracks over Druffel's three strategies (shown in Table 1.5.4) this would seem a promising result. Furthermore since the algorithm tries to force n mappings at the top level it makes no difference which graph appears first and which one appears second.

In Druffel's algorithm this seems to make some difference (shown in Table 1.5.4).

The set of ten 26-point strongly regular graphs behaved in a similar fashion. Over all 45 different pairings of these graphs the algorithm backtracked 26 times in each case at the top level using a mean execution time of 36.30 seconds. Similarily two 40-point 3-strongly regular graphs communicated by R. Mathon required 40 backtracks at the top level to disprove isomorphism in 242.31 seconds. The predicted n^5 times for n=40 based on the 25 and 26 point examples are 332 and 312 seconds respectively. It would seem that these times uphold the theoretical predictions.

Two 27-point strongly regular graphs were tested and the mean execution time to detect non-isomorphism was 72 seconds. This is somewhat higher than would be expected from the twenty-five and twenty-six point examples. The initial type matrix of these graphs is different from the other examples in that it initially has four different integers as opposed to the others which have only three. Perhaps the extra processing due to this would explain the higher computation time. The fact that the initial type matrix is different than for the other 25, 26 and 40 point examples would indicate that the 27-point example has some different properties and so its time is not necessarily the same as for the others.

When each one of the eight 25-point strongly regular graphs was paired with itself, the identity isomorphism was discovered using 0 backtracks in a mean time of 11.54 seconds. Naturally the time varied on the different examples because the number of forced mappings to find the identity is not constant over all examples. Similarily the identity isomorphism was detected between identical copies of the ten 26-point strongly regular graphs using 0 backtracks in a mean time of 14.24 sec.

The algorithm is ordered in such a way that it finds the identity isomorphism right away with no backtracking. However in general for isomorphisms of these graphs the algorithm may have to backtrack up to n-1 times at the top level (depending on how the nodes are re-labeled) until the forced mapping at the top level is successful. Then the method will proceed to fix enough mappings to find the isomorphism. So the maximum number of times the type matrix routine is called upon to determine an isomorphism for these backtracking type graphs is of order $n^2*(n^5 + n^4 + n^2) = n^7 + n^6 + n^4$. Remember that this is based on the assumption that the algorithm will only backtrack at the top level.

5.3 The Automorphism Group Algorithm

It is certain that we cannot claim that the algorithm for finding the generators of the automorphism group will find only the minimum necessary generators of the whole group. The mapping γ_2 in the example of section 4.0 is certainly not needed. However given the nested structure of the stabilizer subgroups of the graph, the algorithm will proceed to look for mappings at each level that contain large orbits and so it will often determine close to the minimum number of generators. It is not very meaningful therefore to try and predict times for finding the generators of the automorphism group. This clearly depends on the number of generators the algorithm finds which in turn depends on the nested structure of the stabilizer subgroups of the graph. In Appendix II however, we present the set of eight strongly regular 25-point graphs with the generators of their automorphism groups.

One criticism that might be launched against the algorithms of this thesis is that they rely on the comparison of real numbers. When comparing real numbers on the computer it is necessary to decide within what small range of difference we would consider two real numbers to be the same. In respect to the algorithms proposed in this thesis the greatest danger would be in considering two numbers the same when in fact they are different. However, if this situation should occur the algorithm would still give the correct answer except that it would back-track more times than necessary in attempting to map the nodes onto each other. To safeguard against "too much" backtracking due to incorrect comparison of real numbers, the program could monitor the numbers and dynamically change the measure if necessary. Certainly in the many graphs that we tried up to as high as 40 points, the comparison of real numbers presented no problem. It is believed that a contributing factor to this is that the eigenvalue and eigenvector routines are very accurate for the type of matrices that we are dealing with.

CHAPTER 6

Conclusion

6.0 General Applicability

So far our attention has been directed at connected graphs without loops or multiple edges. For these graphs a solution is guaranteed and sometimes it will require backtracking. Although it is not certain which general classes of graphs will require backtracking, some strongly regular graphs and graphs of designs are of this nature. If we conjecture that backtracking will only occur at the top level between non-isomorphic graphs then the computation has an upper bound of n^6 to determine non-isomorphism.

Let us now consider for which other types of graphs our algorithms may be employed. If a graph has loops then the elements along the main diagonal of the adjacency matrix may be non-zero but the rest of the matrix is still symmetric. Further if multiple edges are permitted then the symmetry of the matrix is unaltered. Therefore our algorithms may be applied to connected graphs with loops and/or multiple edges.

J. S. Clowes [1977] has suggested that the methods of this thesis may be used for directed graphs if we use Hermitian matrices for the adjacency matrices. A complex number may be used to indicate a directed edge from i to j in the i,j^{th} position of the matrix and its complex conjugate in the j,i^{th} position of the matrix would mean a directed edge from i to j. Hermitian matrices are a more general form of which real symmetric matrices are a special case. Hermitian matrices have all real eigenvalues and their eigenvectors corresponding to distinct eigenvalues are orthogonal. Handling Hermitian matrices on the computer is no great drawback as many languages provide for the use of complex numbers. It is worth noting here that if we do use Hermitian matrices

as our adjacency matrix then our algorithm could handle graphs containing both directed and undirected edges.

6.1 Areas of Investigation For Future Research

It is suggested that future research might be directed at explaining why some graphs have a similar matrix decomposition and yet are not isomorphic. Further it is desirable to be able to state for which set of graphs this occurs. The theory of how the matrix decomposition of the graph is related to the graphical properties needs to be developed in order to make concrete statements about some of these matters. If this theory is developed perhaps a means of discerning the "difference" between these difficult non-isomorphic graphs without backtracking will be possible.

BIBLIOGRAPHY

AHO, A.; HOPCROFT, J.E.; ULLMAN, J.D. [1974] The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

BAKER, G.A. [1966] Drum shapes and isospectral graphs. J. Math. Phys. 7 (1966), 2238-2242.

BAKER, H.H.; DEWDNEY, A.K.; SZILARD, A.L. [1974] Generating the nine-point graphs. Mathematics of Computation 28 #127 (July 1974), 833-838.

BAKHOVSKII, E. [1965] Eigenvalues of adjacency matrices for regular graphs. Sibirskii Matematicheskii Zhurnal 6 #1 (1965), 44-49.

BERTZISS, A.T. [1973] A backtrack procedure for isomorphism of directed graphs. JACM 20 #3 (July 1973), 365-377.

BIGGS, N. [1974] Algebraic Graph Theory. Cambridge Tracts in Mathematics #67, Cambridge University Press, 1974.

BIRKHOFF, G.; MACLANE, S. [1953] A Survey of Modern Algebra. Macmillan, New York, 1953.

BÖHM, C.; SANTOLINI, A. [1964] A quasi-decision algorithm for the P-equivalence of two matrices. I.C.C. Bull. V.3 (1964), 57-69.

BOOTH, K.S. [1976] Problems polynomially equivalent to graph isomorphism. Proceedings of the Symposium on New Directions and Recent Results in Algorithms and Complexity, Carnegie-Mellon University, (April 1976).

BOSE, R.C. [1961] At the meeting of the American Mathematical Society, April 6, 1961.

BOSE, R.C. [1963] Strongly regular graphs, partial geometries, and partially balanced designs. Pacific J. Mathematics 13 (1963), 389-419.

BRUCK, R.H. [1963] Finite nets II: uniqueness and imbedding. Pacific J. Mathematics 13 (1963), 421-457.

CHAO, C.Y. [1965] On groups and graphs. Proc. Amer. Math. Soc. 118 (1965), 488-497.

CLOWES, J.S. [1977] Private communication. Computing Laboratory, The University of Newcastle upon Tyne, 1977.

COLLATZ, L.; SINOGOWITZ, U. [1957] Spektren endlicher graphen. Abh. Math. Sem. University Hamburg 21 (1957), 63-67.

COOK, S.A. [1970] The complexity of theorem-proving procedures. Proc. of the Third Annual ACM Symposium on the Theory of Computing, (1970), 151-158.

COOK, S.A. [1972] Linear time simulation of deterministic two-way pushdown automata. IFIP CONGRESS 71: Foundations of Information Processing, Proceedings Ljubljana, August 1971, North-Holland Pub. Co., Amsterdam, (1972), 174-179.

COOK, S.A.; RECKHOW, R.A. [1973] Time-bounded random access machines. J. Computer and System Sciences 7 #4 (1973), 354-375.

CORNEIL, D.G. [1968] Graph Isomorphism. Ph.D. Thesis, University of Toronto, Dept. of Computer Science, 1968.

CORNEIL, D.G.; GOTLIEB, C.C. [1970] An efficient algorithm for graph isomorphism. JACM 17 #1 (1970), 51-64.

CORNEIL, D.G. [1974] The analysis of graph theoretical algorithms. U. of Toronto, Dept. of Computer Science, Technical Report #74, (December 1974).

CORNEIL, D.G.; READ, R.C. [1975] The graph isomorphism disease. Submitted for publication.

CVETKOVIĆ, D. [1971 a] Graphs and their Spectra. Doctoral Thesis, U. of Belgrade, 1971.

CVETKOVIĆ, D. [1971 b] Graphs and their Spectra (an abridged version of the thesis). Publications of the Faculty of Electrical Engineering of the University of Belgrade #354 (1971), 1-50.

DOOB, M. [1970] Graphs with a small number of distinct eigenvalues. Ann. New York Acad. Sci. 175 #1 (1970), 104-110.

DRUFFEL, L.E. [1975] Graph related algorithms: isomorphism, automorphism, and containment. Tech. Report 75-1, Systems and Information Science, School of Engineering, Vanderbilt University (1975).

DRUFFEL, L.E.; SCHMIDT, D.C. [1976] A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. JACM 23 #3 (July 1976), 433-445.

FISCHER, M. [1966] On hearing the shape of a drum. J. Comb. Theory 1 (1966), 105-125.

FLOYD, R.W. [1962] Algorithm 97, shortest path. CACM 5 #6 (June 1962), 345.

FRUCHT, R. [1949] Graphs of degree three with a given abstract group. Can. J. Math. 1 (1949), 365-378.

GANTMACHER, F.R. [1960] The Theory of Matrices Vol. II (English translation), Chelsea Publishing Co., New York, 1960.

GELBART, R. [1976] Use of the spectrum in graph isomorphism. M.Sc. Thesis, Dept. of Computer Science, U. of British Columbia, (Feb. 1976).

GIBBONS, P.B. [1976] Computing techniques for the construction and analysis of block designs. Ph.D. thesis, Dept. of Computer Science, U. of Toronto, (April 1976). Available as Technical Report #92, May 1976.

GOURLAY, A.R.; WATSON, G.A. [1973] Computational Methods for Matrix Eigenproblems. John Wiley & Sons, London 1973.

HARARY, F. [1955] The number of linear, directed, rooted, and connected graphs. Trans. Amer. Math. Soc. 78 (1955), 445-463.

HARARY, F. [1969] Graph Theory. Addison-Wesley, Reading, 1969.

HARARY, F.; KING, C.; MOWSHOWITZ, A.; READ, R. [1971] Cospectral graphs and digraphs. Bull. London Math. Soc. 3 #9 (Nov. 1971), 321-328.

HOFFMAN, A. [1963] On the polynomial of a graph. Amer. Math. Monthly 70 (1963), 30-36.

HOFFMAN, A. [1969] The eigenvalues of the adjacency matrix of a graph. Combinatorial mathematics and its applications, (1969), U. of North Carolina Press, Chapel Hill, 578-584.

HOFFMAN, A. [1974] Eigenvalues of graphs. IBM Watson Research Centre, Yorktown Heights, New York, report #RC4688 (#20895), Jan. 18, 1974.

HOPCROFT, J.; TARJAN, R. [1971] A V^2 algorithm for determining isomorphism of planar graphs. Information Processing Letters 1 #1 (Feb. 1971), 32-34.

HOPCROFT, J.; TARJAN, R. [1972] Isomorphism of planar graphs (working paper). Complexity of Computer Computations, (R.E. Miller & J.W. Thatcher (eds.)), Plenum Press, New York, (1972), 143-150.

HOPCROFT, J.; TARJAN, R. [1973] A V log V algorithm for isomorphism of tri-connected planar graphs. JCSS 7 #3 (1973), 323-331.

HOPCROFT, J.; WONG, J. [1974] Linear time algorithm for isomorphism of planar graphs (extended abstract). Sixth Annual ACM Symposium on Theory of Computing, Seattle, May 1974.

IZBICKI, H. [1960] Reguläre Graphen beliebigen Grades mit vorgegebenen Eigenschaften. Monatshefte für Math. 64 (1960), 15-21.

KARP, R.M. [1972] Reducibility among combinatorial problems. Complexity of Computer Computations, (R.E. Miller & J.W. Thatcher (eds.)), Plenum Press, (1972), 85-103.

KIRKPATRICK, D. [1974] Topics in the complexity of combinatorial algorithms. Ph.D. thesis, U. of Toronto, Dept. of Computer Science, (1974). Available as technical report #74, December 1974.

KNUTH, D.E. [1973] The Art of Computer Programming Vol. 3, Sorting and Searching. Addison-Wesley, 1973.

KNUTH, D.E. [1975] Estimating the efficiency of backtrack programs. Math. Comp. 29 #129 (1975), 121-136.

KRIDER, L. [1964] A flow analysis algorithm. JACM 11 #4 (Oct. 1964), 429-436.

KUHN, W.W. [1971 a] Graph isomorphism using vertex adjacency matrices. Proc. 25th Summer Meeting Canadian Math. Congress (1971), Lakehead University, 471-476.

KUHN, W.W. [1971 b] An algorithm for graph isomorphism using vertex adjacency matrices. Dissertation, Dept. of Applied Mathematics, U. of Pennsylvania, (1971).

LEVI, G. [1974] Graph Isomorphism: A Heuristic Edge-Partitioning-Oriented Algorithm. Computing 12 (1974), 291-313.

MCKAY, B. [1976 a] Backtrack programming and the graph isomorphism problem. M.Sc. thesis, Dept. of Mathematics, U. of Melbourne, July 1976.

MCKAY, B. [1976 b] Backtrack programming and isomorph rejection on ordered subsets. Dept. of Mathematics, Research Report No. 45 (1976), U. of Melbourne.

MARCUS, M.; MINC, H. [1964] A Survey of Matrix Theory and Matrix Inequalities. Allyn and Bacon, Rockleigh, N.J., 1964.

MATHON, R. [1975] reported in GIBBONS [1976].

MOWSHOWITZ, A. [1969] The group of a graph whose adjacency matrix has all distinct eigenvalues. Proof Techniques in Graph Theory, ed. F. Harary, Academic Press, New York, (1969), 109-110.

MOWSHOWITZ, A. [1972] The characteristic polynomial of a graph. J. of Comb. Theory 12 #2 (April 1972), 177-193.

MOWSHOWITZ, A. [1973] The adjacency matrix and the group of a graph. New Directions in the Theory of Graphs (F. Harary ed.), Academic Press, N.Y.-London, (1973), 129-148.

MOWSHOWITZ, A. [1974] Private communication, Dept. of Computer Science, U. of British Columbia, April 1974.

NAGLE, J.F. [1966] On ordering and identifying undirected linear graphs. J. Math. Phys. 7 #9 (1966), 1588-1592.

PETERSDORF, M.; SACHS, H. [1969] Spektrum und Automorphismengruppe eines Graphen. Combinatorial Theory and its Applications III, North-Holland, Amsterdam, (1969), 891-907.

PÓLYA, G. [1937] Kombinatorische Anzahlbestimmungen für Gruppen, Graphen, und chemische Verbindungen. Acta. Math. 68 (1937), 145-254.

RANDIC, M. [1974] On the recognition of identical graphs representing molecular topology. J. Chem. Phys. 60 (1974), 3920-3928.

REINSCH, C.; WILKINSON, J.H. [1971] Linear Algebra, (Vol. II of Handbook for Automatic Computation), Springer-Verlager, Berlin, Heidelberg, New York, 1971.

REISCHER, C.; SIMOVICI, D. [1971] On the matrix equation $PAP^T = B$ in the two-element Boolean Algebra and some applications in Graph Theory. An. STI University "Al. I. Cuza", Iaşi Sect. I a Mat. (N.S.) 17 (1971), 263-274.

SAUCIER, G. [1971] Un algorithme efficace recherchant l'isomorphisme de 2 graphes. Revue Française Informatique Recherche Opérationnelle 5 Sér R-3 (1971), 39-51.

SEIDEL, J. [1968] Strongly regular graphs with (-1,1,0) adjacency matrix having eigenvalue 3. Linear Algebra and Appl. 1 (1968), 281-298.

SENETA, F. [1973] Non-Negative Matrices. George Allen & Unwin Ltd., London, 1973.

SHAH, Y.J.; DAVIDA, G.I.; MCCARTHY, M.K. [1974] Optimum features and graph isomorphism. IEEE Transactions on Systems, Man, and Cybernetics SMC-4 #3 (May 1974), 313-319.

SIROVICH, F. [1971] Isomorfismo fra grafi: un algoritmo efficiente per trovare tutti gli isomorfismi. Calcolo 8 (1971), 301-337.

SNOW, C.R. [1973] An analysis of the structure of trees and graphs. Ph.D. Thesis, (1973), Computing Laboratory, University of Newcastle upon Tyne.

STOCKTON, F.G. [1968] Linearization and standardization of graphs. Shell Development Co., Technical Progress Report No. 2-68. Project No. 34430, 1968.

SUSSENGUTH, E.H. [1965] A graph-theoretic algorithm for matching chemical structures. J. Chem. Doc. 5 (1965), 36-43.

TURING, A.M. [1936] On computable numbers with and application to the Entscheidungsproblem. Proc. London Math. Soc. Ser. 2 42 (1936), 230-265. Corrections, ibid. 43 (1937), 544-546.

TURNER, J. [1967] Some isomorphism and classification problems in graph theory. Stanford University, Dept of Math., Ph.D. thesis (1967).

TURNER, J. [1968] Generalized matrix functions and the graph isomorphism problem. SIAM J. Appl. Math. 16 #3 (May 1968).

UNGER, S.H. [1964] GIT - a heuristic program for testing pairs of directed line graphs for isomorphism. CACM 7 #1 (Jan. 1964), 26-34.

VAN DER WAERDEN, B.L. [1970] Algebra (vol.1). Frederick Ungar Publishing Co., New York, 1970.

WEINBERG, L. [1965] Plane representations and codes for planar graphs. Proc. Third Allerton Conf. on Circuit and System Theory, Univ. of Illinois, (Oct. 1965), 733-744.

WEINBERG, L. [1966] A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. IEEE Trans. on Circuit Theory 13 (1966), 142-148.

WILF, H. [1967] The eigenvalues of a graph and its chromatic number. J. London Math. Soc. 42 (1967), 330-332.

WILKINSON, J.H. [1965] The Algebraic Eigenvalue Problem. Oxford University Press, London, 1965.

APPENDIX I

The Eigenvalues, Eigenvectors, and H_1 Matrices For Various Cospectral Graphs

EIGENVALUE: -0.20843E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.19190E+00
-0.19190E+00
0.39992E+00
-0.44962E+00
0.53707E+00
-0.33481E+00
0.16066E+00
-0.33481E+00
0.16055E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.23674E+00
-0.23674E+00
0.19337E+00
-0.55467E+00
0.26616E+00
0.39639E+00
-0.27139E+00
0.16919E+00
-0.31185E-01

EIGENVALUE: -0.15713E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.36260E+00
-0.36260E+00
0.55394E+00
-0.17067E+00
-0.30153E+00
0.32243E+00
-0.20513E+00
0.32243E+00
-0.20513E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.22799E+00
-0.22799E+00
0.35837E+00
-0.10731E+00
0.68272E-01
-0.25795E+00
0.51279E+00
-0.54805E+00
0.34867E+00

EIGENVALUE: -0.10000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.89500E-16
0.27104E-15
-0.30903E-15
-0.12795E-16
0.21093E-15
-0.50000E+00
0.50000E+00
0.50000E+00
-0.50000E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.35355E+00
-0.35355E+00
0.35355E+00
0.35355E+00
-0.35355E+00
-0.35355E+00
-0.31191E-15
0.35355E+00
-0.35355E+00

EIGENVALUE: -0.43173E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.28583E+00
-0.28583E+00
0.12390E+00
0.51838E+00
-0.38720E+00
-0.18424E+00
0.42573E+00
-0.18424E+00
0.42573E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.13028E+00
-0.13028E+00
0.56245E-01
0.23627E+00
-0.54726E+00
0.38901E+00
-0.40322E+00
-0.21450E+00
0.49683E+00

EIGENVALUE: 0.0

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.70711E+00
-0.70711E+00
0.0
-0.58878E-16
0.0
0.58878E-16
0.0
0.0
0.0

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.70711E+00
-0.70711E+00
-0.29439E-16
0.10486E-31
0.29439E-16
-0.12925E-63
-0.10486E-31
0.64626E-64
0.10486E-31

EIGENVALUE: 0.43173E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.28583E+00
0.28583E+00
0.12340E+00
-0.51838E+00
-0.34720E+00
0.18424E+00
0.42674E+00
0.18424E+00
0.42674E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.13028E+00
-0.13028E+00
-0.56245E-01
0.23627E+00
0.54726E+00
-0.38901E+00
-0.40422E+00
0.21450E+00
0.49683E+00

EIGENVALUE: 0.10003E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.41603E-15
-0.61292E-15
-0.73051E-15
0.99560E-16
0.17463E-15
0.50000E+00
0.50000E+00
-0.50000E+00
-0.50000E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.35355E+00
-0.35355E+00
-0.35355E+00
0.35355E+00
0.35355E+00
0.35355E+00
-0.10035E-15
-0.35355E+00
-0.35355E+00

EIGENVALUE: 0.15718E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.36260E+00
-0.36260E+00
-0.56994E+00
-0.17067E+00
0.30168E+00
0.32243E+00
0.20513E+00
0.32243E+00
0.20513E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.22799E+00
0.22799E+00
0.35837E+00
0.10731E+00
0.68272E-01
-0.25796E+00
-0.51279E+00
-0.54806E+00
-0.34867E+00

EIGENVALUE: 0.20840E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.19190E+00
0.19190E+00
0.39992E+00
0.44962E+00
0.53707E+00
0.33481E+00
0.16066E+00
0.33481E+00
0.16066E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.23674E+00
0.23674E+00
0.49337E+00
0.55467E+00
0.26616E+00
0.39639E+00
0.27139E+00
0.16919E+00
0.81185E-01

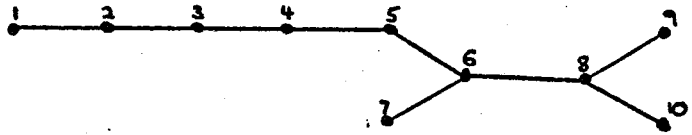
TWO COSPECTRAL GRAPHS

THERE ARE 2 COSPECTRAL GRAPHS ON 10 POINTS IN THIS EXAMPLE

ADJACENCY LISTS

G1

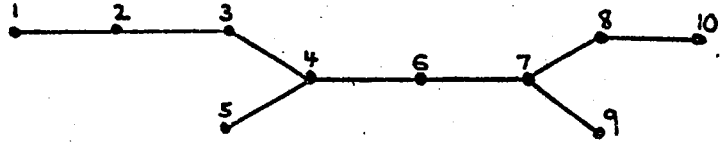
```
 1:  2  0
 2:  3  0
 3:  4  0
 4:  5  0
 5:  6  0
 6:  7  8  0
 7:  0
 8:  9 10  0
 9:  0
10:  0
```



Two Non-Isomorphic Trees

G2

```
 1:  2  0
 2:  3  0
 3:  4  0
 4:  5  6  0
 5:  0
 6:  7  0
 7:  8  9  0
 8: 10  0
 9:  0
10:  0
```



THE EIGENVALUES ARE:

-0.20886E+01 -0.16810E+01 -0.11491E+01 -0.70108E+00 -0.50675E-16 0.50675E-16 0.70108E+00 0.11491E+01 0.16810E+01 0.20886E+01

-0.20886E+01 -0.16810E+01 -0.11491E+01 -0.70108E+00 -0.24145E-15 0.92410E-16 0.70108E+00 0.11491E+01 0.16810E+01 0.20886E+01

MULTIPLICITY OF THE EIGENVALUES: 1 1 1 1 2 1 1 1 1

EIGENVALUE: -0.20886E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.57158E-01
0.11938E+00
-0.19219E+00
0.28202E+00
-0.39685E+00
0.54685E+00
-0.26182E+00
-0.48349E+00
0.23149E+00
0.23149E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.97990E-01
-0.20465E+00
0.32948E+00
-0.48349E+00
0.23149E+00
0.49886E+00
-0.45401E+00
0.28202E+00
0.21737E+00
-0.13503E+00

EIGENVALUE: -0.16810E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.29168E+00
-0.49031E+00
0.53254E+00
-0.40489E+00
0.14809E+00
0.15596E+00
-0.92776E-01
-0.31748E+00
0.18886E+00
0.18886E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.22870E+00
-0.38445E+00
0.41756E+00
-0.31748E+00
0.18886E+00
-0.72746E-01
0.43976E+00
-0.40489E+00
-0.26161E+00
0.24086E+00

EIGENVALUE: -0.11491E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.35899E+00
0.40790E+00
-0.11372E+00
-0.27723E+00
0.43228E+00
-0.21949E+00
0.19101E+00
-0.37108E+00
0.32294E+00
0.32294E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.47517E+00
-0.54600E+00
0.15222E+00
0.37108E+00
-0.32294E+00
-0.25568E+00
-0.77290E-01
0.27723E+00
0.67264E-01
-0.24126E+00

EIGENVALUE: -0.70108E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.40080E+00
0.28100E+00
0.20380E+00
-0.42388E+00
0.93373E-01
0.35842E+00
-0.51123E+00
0.16658E+00
-0.23760E+00
-0.23760E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.15751E+00
0.11043E+00
0.80091E-01
-0.16658E+00
0.23760E+00
-0.20091E+00
0.30743E+00
0.42388E+00
-0.43851E+00
-0.60860E+00

EIGENVALUE: -0.50675E-16

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.35355E+00	0.35355E+00
-0.19626E-16	0.19626E-16
-0.35355E+00	-0.35355E+00
0.98131E-16	-0.98131E-16
0.35355E+00	0.35355E+00
-0.53972E-16	0.53972E-16
-0.35355E+00	-0.35355E+00
0.25337E-16	0.25337E-16
0.50000E+00	-0.50000E+00
-0.50000E+00	0.50000E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.41617E+00	0.44923E+00
0.85278E-16	-0.79469E-17
0.41617E+00	-0.44923E+00
-0.97478E-16	-0.45759E-16
0.14609E+00	0.69185E+00
-0.56226E+00	-0.24263E+00
0.10007E-15	0.44057E-16
0.24162E-31	-0.40713E-32
0.56226E+00	0.24263E+00
-0.10007E-15	-0.44057E-16

MULTIPLE EIGENVALUE: -0.50675E-16

CORRESPONDING H(I) MATRIX FOR G1:

0.2500E+00	-0.1362E-32	-0.2500E+00	0.3541E-32	0.2500E+00	-0.2656E-32	-0.2500E+00	0.1792E-16	0.1507E-16	-0.1507E-16
-0.1362E-32	0.7704E-33	0.8171E-33	-0.3852E-32	-0.1089E-32	0.2119E-32	0.8171E-33	0.6653E-48	-0.1963E-16	0.1963E-16
-0.2500E+00	0.8171E-33	0.2500E+00	-0.8171E-33	-0.2500E+00	0.1158E-32	0.2500E+00	-0.1792E-16	-0.1191E-17	0.1191E-17
0.3541E-32	-0.3852E-32	-0.8171E-33	0.1926E-31	0.2179E-32	-0.1059E-31	-0.8171E-33	-0.3561E-47	0.9813E-16	-0.9813E-16
0.2500E+00	-0.1089E-32	-0.2500E+00	0.2179E-32	0.2500E+00	-0.1907E-32	-0.2500E+00	0.1792E-16	0.8129E-17	-0.8129E-17
-0.2656E-32	0.2119E-32	0.1158E-32	-0.1059E-31	-0.1907E-32	0.5826E-32	0.1158E-32	0.1908E-47	-0.5397E-16	0.5397E-16
-0.2500E+00	0.8171E-33	0.2500E+00	-0.8171E-33	-0.2500E+00	0.1158E-32	0.2500E+00	-0.1792E-16	-0.1191E-17	0.1191E-17
0.1792E-16	0.6653E-48	-0.1792E-16	-0.3561E-47	0.1792E-16	0.1908E-47	-0.1792E-16	0.1284E-32	-0.1835E-31	0.1835E-31
0.1507E-16	-0.1963E-16	-0.1191E-17	0.9813E-16	0.8129E-17	-0.5397E-16	-0.1191E-17	-0.1835E-31	0.5000E+00	-0.5000E+00
-0.1507E-16	0.1963E-16	0.1191E-17	-0.9813E-16	-0.8129E-17	0.5397E-16	0.1191E-17	0.1835E-31	-0.5000E+00	0.5000E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.3750E+00	-0.3905E-15	-0.3750E+00	0.2001E-16	0.2500E+00	0.1250E+00	-0.2185E-16	-0.1188E-31	-0.1250E+00	0.2185E-16
-0.3906E-16	0.7335E-32	0.3906E-16	-0.7949E-32	0.6960E-17	-0.4602E-16	0.8183E-32	0.2093E-47	0.4602E-16	-0.8183E-32
-0.3750E+00	0.3905E-15	0.3750E+00	-0.2001E-16	-0.2500E+00	-0.1250E+00	0.2185E-16	0.1188E-31	0.1250E+00	-0.2185E-16
0.2001E-16	-0.7949E-32	-0.2001E-16	0.1160E-31	-0.4590E-16	0.6591E-16	-0.1177E-31	-0.2169E-47	-0.6591E-16	0.1177E-31
0.2500E+00	0.6960E-17	-0.2500E+00	-0.4590E-16	0.5000E+00	-0.2500E+00	0.4510E-16	0.7131E-33	0.2500E+00	-0.4510E-16
0.1250E+00	-0.4602E-16	-0.1250E+00	0.6591E-16	-0.2500E+00	0.3750E+00	-0.6695E-16	-0.1260E-31	-0.3750E+00	0.6695E-16
-0.2185E-16	0.8183E-32	0.2185E-16	-0.1177E-31	0.4510E-16	-0.6695E-16	0.1195E-31	0.2239E-47	0.6695E-16	-0.1195E-31
-0.1188E-31	0.2093E-47	0.1188E-31	-0.2169E-47	0.7131E-33	-0.1260E-31	0.2239E-47	0.6004E-63	0.1260E-31	-0.2239E-47
-0.1250E+00	0.4602E-16	0.1250E+00	-0.6591E-16	0.2500E+00	-0.3750E+00	0.6695E-16	0.1260E-31	0.3750E+00	-0.6695E-16
0.2185E-16	-0.8183E-32	-0.2185E-16	0.1177E-31	-0.4510E-16	0.6695E-16	-0.1195E-31	-0.2239E-47	-0.6695E-16	0.1195E-31

EIGENVALUE: 0.70109E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.30080E+00
0.28100E+00
-0.20380E+00
-0.42388E+00
-0.93371E-01
0.35842E+00
0.51123E+00
-0.16658E+00
-0.23760E+00
-0.23760E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.15751E+00
0.11043E+00
-0.80091E-01
-0.16658E+00
-0.23760E+00
0.20091E+00
0.30743E+00
-0.42388E+00
0.43851E+00
-0.60460E+00

EIGENVALUE: 0.11491E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.35899E+00
0.40790E+00
0.11372E+00
-0.27723E+00
-0.43220E+00
-0.21949E+00
-0.19101E+00
0.37108E+00
0.32294E+00
0.32294E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.47517E+00
-0.54600E+00
-0.15222E+00
0.37108E+00
0.32294E+00
0.25568E+00
-0.77290E-01
-0.27723E+00
-0.67264E-01
-0.24126E+00

EIGENVALUE: 0.16810E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.29168E+00
0.49031E+00
0.53254E+00
0.40489E+00
0.14809E+00
-0.15596E+00
-0.92776E-01
-0.31748E+00
-0.18886E+00
-0.18886E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.22870E+00
0.38485E+00
0.41756E+00
0.31748E+00
0.18886E+00
-0.72746E-01
-0.43976E+00
-0.40489E+00
-0.26161E+00
-0.24086E+00

EIGENVALUE: 0.20886E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.57158E-01
-0.11938E+00
-0.19219E+00
-0.28202E+00
-0.19685E+00
-0.54685E+00
-0.26182E+00
-0.48349E+00
-0.23149E+00
-0.23149E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.97990E-01
0.20466E+00
0.32948E+00
0.48349E+00
0.23149E+00
0.44886E+00
0.45401E+00
0.28202E+00
0.21737E+00
0.13503E+00

TWO COSPECTRAL GRAPHS

THERE ARE 2 COSPECTRAL GRAPHS ON 10 POINTS IN THIS EXAMPLE

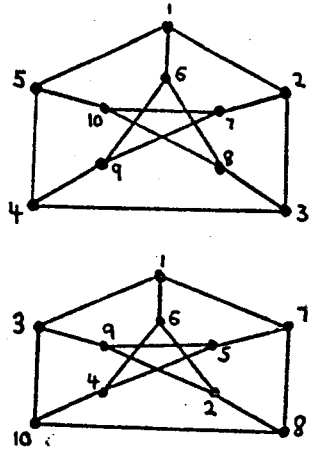
ADJACENCY LISTS

G1

```
 1:  2  5  6  0
 2:  3  7  0
 3:  4  8  0
 4:  5  9  0
 5: 10  0
 6:  8  9  0
 7:  9 10  0
 8: 10  0
 9:  0
10:  0
```

G2

```
 1:  3  6  7  0
 2:  6  8  9  0
 3:  9 10  0
 4:  5  6 10  0
 5:  7  9  0
 6:  0
 7:  8  0
 8: 10  0
 9:  0
10:  0
```



Two Isomorphic Petersen Graphs

THE EIGENVALUES ARE:

-0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 0.10000E+01 0.10000E+01 0.10000E+01 0.10000E+01 0.10000E+01 0.30000E+01

-0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 0.10000E+01 0.10000E+01 0.10000E+01 0.10000E+01 0.10000E+01 0.30000E+01

MULTIPLICITY OF THE EIGENVALUES: 4 5 1

EIGENVALUE: -0.20000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.33694E+00	0.51004E+00	-0.15577E+00	0.45487E-01
-0.10616E+00	-0.61451E+00	-0.87620E-01	-0.58608E-01
0.46526E+00	0.37134E+00	-0.18997E+00	0.97719E-01
-0.38126E+00	-0.23706E-01	0.18180E+00	-0.47013E+00
0.35910E+00	-0.24316E+00	0.25155E+00	0.38554E+00
0.42095E+00	-0.16240E+00	0.14761E+00	-0.41790E+00
0.84008E-01	0.34764E+00	0.52098E+00	-0.25990E-01
-0.44310E+00	-0.10447E+00	0.28575E+00	0.33330E+00
-0.61845E-01	-0.80764E-01	-0.42519E+00	0.45701E+00
-0.69958E-16	0.28896E-16	-0.52914E+00	-0.34643E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.10541E+00	0.32961E+00	0.75786E-01	-0.52393E+00
0.10541E+00	0.37451E+00	0.45853E+00	0.19590E+00
-0.42164E+00	0.69713E-01	-0.13722E-01	0.46602E+00
-0.42164E+00	-0.94526E-01	0.41019E+00	-0.21221E+00
0.10541E+00	0.49385E+00	-0.34812E+00	0.15430E+00
0.10541E+00	-0.30480E+00	-0.47225E+00	0.27012E+00
0.10541E+00	-0.42414E+00	0.33440E+00	0.31172E+00
-0.42164E+00	0.24813E-01	-0.39647E+00	-0.25381E+00
0.10541E+00	-0.46904E+00	-0.48342E-01	-0.40811E+00
0.63246E+00	0.0	0.0	0.0

MULTIPLE EIGENVALUE: -0.20000E+01

CORRESPONDING H(I) MATRIX FOR G1:

0.4000E+00	-0.2667E+00	0.6667E-01	0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01
-0.2667E+00	0.4000E+00	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01
0.6667E-01	-0.2667E+00	0.4000E+00	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	0.6667E-01
0.6667E-01	0.6667E-01	-0.2667E+00	0.4000E+00	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01
-0.2667E+00	0.6667E-01	0.6667E-01	-0.2667E+00	0.4000E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00
-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01	0.4000E+00	0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01
0.6667E-01	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01	0.4000E+00	0.6667E-01	-0.2667E+00	-0.2667E+00
0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	0.4000E+00	0.6667E-01	-0.2667E+00
0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01	0.4000E+00	0.6667E-01
0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01	0.4000E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.4000E+00	0.6667E-01	-0.2667E+00	0.6667E-01	0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01
0.6667E-01	0.4000E+00	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01
-0.2667E+00	0.6667E-01	0.4000E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	-0.2667E+00
0.6667E-01	0.6667E-01	0.6667E-01	0.4000E+00	-0.2667E+00	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00
0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.4000E+00	0.6667E-01	-0.2667E+00	0.6667E-01	-0.2667E+00	0.6667E-01
-0.2667E+00	-0.2667E+00	0.6667E-01	-0.2667E+00	0.6667E-01	0.4000E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01
-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	0.4000E+00	-0.2667E+00	0.6667E-01	0.6667E-01
0.6667E-01	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.4000E+00	0.6667E-01	-0.2667E+00
0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	0.4000E+00	0.6667E-01
0.6667E-01	0.6667E-01	-0.2667E+00	-0.2667E+00	0.6667E-01	0.6667E-01	0.6667E-01	-0.2667E+00	0.6667E-01	0.4000E+00

EIGENVALUE: 0.10000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.27418E+00	-0.18857E+00	-0.20867E+00	-0.31792E+00	-0.49468E+00
0.17545E+00	-0.57417E+00	0.25599E+00	-0.18043E+00	-0.23303E+00
0.64067E+00	-0.71893E-01	0.49245E-01	-0.24591E+00	0.14656E+00
0.17117E+00	-0.63305E-01	-0.48211E+00	-0.21665E-01	0.48352E+00
-0.55933E-01	0.29043E-01	-0.61115E+00	0.26454E+00	-0.22922E+00
-0.39365E+00	0.35656E+00	0.14650E+00	-0.44203E+00	-0.32390E-01
-0.19108E+00	-0.31371E+00	0.41541E+00	0.42341E+00	0.11505E+00
0.29405E+00	0.56558E+00	0.27537E+00	-0.83821E-01	-0.10393E+00
-0.41356E+00	-0.20455E-01	0.79790E-01	-0.40289E-01	0.56618E+00
0.47035E-01	0.28092E+00	0.79632E-01	0.60412E+00	-0.21810E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.24112E+00	-0.26321E+00	0.55806E+00	0.23570E+00	0.74797E-01
0.26039E-01	0.35179E-01	-0.31428E+00	0.23570E+00	-0.58631E+00
0.52360E+00	0.28007E+00	0.29446E+00	-0.23570E+00	0.71730E-01
-0.61679E+00	0.18558E+00	0.14189E+00	-0.23570E+00	0.97195E-01
-0.26715E+00	0.22803E+00	-0.24378E+00	0.23570E+00	0.51151E+00
-0.34963E+00	-0.42452E-01	0.38567E+00	0.23570E+00	-0.41832E+00
0.67147E-01	-0.50083E+00	-0.12207E+00	0.23570E+00	0.41738E+00
0.93187E-01	-0.46565E+00	-0.43635E+00	-0.23570E+00	-0.16893E+00
0.28248E+00	0.54328E+00	-0.26360E+00	0.23570E+00	-0.30674E-02
0.0	0.0	0.0	-0.70711E+00	0.0

MULTIPLE EIGENVALUE: 0.10000E+01

CORRESPONDING H(I) MATRIX FOR G1:

0.5000E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00
0.1667E+00	0.5000E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00
-0.1667E+00	0.1667E+00	0.5000E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00
-0.1667E+00	-0.1667E+00	0.1667E+00	0.5000E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00
0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	0.5000E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00
-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.5000E+00	-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00
-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.5000E+00	-0.1667E+00	0.1667E+00	0.1667E+00
-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	0.5000E+00	-0.1667E+00	0.1667E+00
-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00	0.5000E+00	-0.1667E+00
-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00	0.5000E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.5000E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00
-0.1667E+00	0.5000E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00
0.1667E+00	-0.1667E+00	0.5000E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	0.1667E+00
-0.1667E+00	-0.1667E+00	-0.1667E+00	0.5000E+00	0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00
-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	0.5000E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00
0.1667E+00	0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	0.5000E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00
0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	0.5000E+00	0.1667E+00	-0.1667E+00	-0.1667E+00
-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	0.5000E+00	-0.1667E+00	0.1667E+00
-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.5000E+00	-0.1667E+00
-0.1667E+00	-0.1667E+00	0.1667E+00	0.1667E+00	-0.1667E+00	-0.1667E+00	-0.1667E+00	0.1667E+00	-0.1667E+00	0.5000E+00

EIGENVALUE: 0.30000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

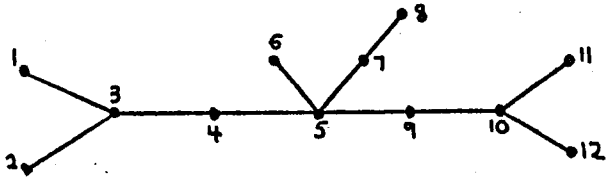
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00
0.31623E+00

TWO COSPECTRAL GRAPHS
THERE ARE 2 COSPECTRAL GRAPHS ON 12 POINTS IN THIS EXAMPLE

ADJACENCY LISTS

G1

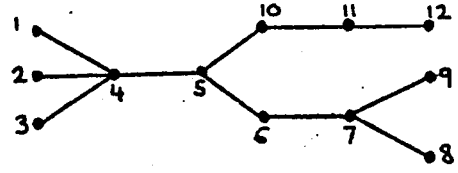
```
 1:  3  0
 2:  3  0
 3:  4  0
 4:  5  0
 5:  6  7  9  0
 6:  0
 7:  8  0
 8:  0
 9: 10  0
10: 11 12  0
11:  0
12:  0
```



Two Non-Isomorphic Trees

G2

```
 1:  4  0
 2:  4  0
 3:  4  0
 4:  5  0
 5:  6 10  0
 6:  7  0
 7:  8  9  0
 8:  0
 9:  0
10: 11  0
11: 12  0
12:  0
```



THE EIGENVALUES ARE:

-0.22725E+01 -0.17321E+01 -0.14924E+01 -0.78014E+00 -0.20637E-15 0.15407E-32 0.15886E-16 0.39578E-16 0.78014E+00 0.14924E+01
0.17321E+01 0.22725E+01

-0.22725E+01 -0.17321E+01 -0.14924E+01 -0.78014E+00 -0.86057E-16 -0.12326E-31 0.0 0.22580E-46 0.78014E+00 0.14924E+01
0.17321E+01 0.22725E+01

MULTIPLICITY OF THE EIGENVALUES: 1 1 1 1 4 1 1 1 1

EIGENVALUE: -0.22725E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.11799E+00
0.11799E+00
-0.26814E+00
0.37336E+00
-0.58033E+00
0.25537E+00
0.31669E+00
-0.13936E+00
0.37336E+00
-0.26814E+00
0.11799E+00
0.11799E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.23374E+00
0.23374E+00
0.23374E+00
-0.53117E+00
0.50587E+00
-0.32546E+00
0.23374E+00
-0.10286E+00
-0.10286E+00
-0.29296E+00
0.15987E+00
-0.70350E-01

EIGENVALUE: -0.14924E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.27383E+00
-0.27383E+00
0.40866E+00
-0.62197E-01
-0.31584E+00
0.21164E+00
0.38410E+00
-0.25738E+00
-0.62197E-01
0.40866E+00
-0.27383E+00
-0.27383E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.17581E+00
0.17581E+00
0.17581E+00
-0.26238E+00
-0.13588E+00
-0.26759E-01
0.17581E+00
-0.11781E+00
-0.11781E+00
0.49192E+00
-0.59824E+00
0.40087E+00

EIGENVALUE: -0.17321E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.28868E+00
0.28868E+00
-0.50000E+00
0.28868E+00
0.35328E-16
-0.43163E-16
-0.13962E-15
0.44297E-15
-0.28868E+00
0.50000E+00
-0.28868E+00
-0.28868E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.20412E+00
0.20412E+00
0.20412E+00
-0.35355E+00
-0.35691E-15
0.35355E+00
-0.61237E+00
0.35355E+00
0.35355E+00
0.41215E-15
-0.35693E-15
0.20507E-15

EIGENVALUE: -0.78019E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.13504E+00
-0.13504E+00
0.10535E+00
0.18789E+00
-0.25193E+00
0.32293E+00
-0.50217E+00
0.64369E+00
0.18789E+00
0.10535E+00
-0.13504E+00
-0.13504E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.19863E+00
-0.19863E+00
-0.19863E+00
0.15496E+00
0.47501E+00
-0.35426E+00
-0.19863E+00
0.25461E+00
0.25461E+00
-0.17127E+00
-0.34139E+00
0.43760E+00

EIGENVALUE: -0.20637E-15

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.99592E-01	0.70711E+00	0.12608E+00	-0.30534E+00
-0.99592E-01	-0.70711E+00	0.12608E+00	-0.30534E+00
0.34627E-16	-0.31153E-20	0.44781E-17	0.66314E-17
0.19918E+00	-0.35693E-16	-0.25216E+00	0.61068E+00
-0.35109E-16	0.22742E-33	0.15709E-16	0.59511E-17
0.28653E+00	0.40388E-16	-0.23567E+00	-0.65863E+00
-0.32284E-17	-0.82716E-17	-0.12525E-15	0.94889E-16
0.34023E-16	0.31153E-20	0.27719E-16	0.87729E-17
-0.48571E+00	0.35767E-17	0.48783E+00	0.47908E-01
0.53533E-16	-0.12062E-49	-0.11755E-16	-0.25718E-17
0.74511E+00	0.42522E-17	0.25213E+00	0.17032E-01
-0.25940E+00	-0.78288E-17	-0.73996E+00	-0.64981E-01

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.13137E-15	0.40825E+00	0.70544E+00	-0.48549E-01
-0.10675E-15	0.40825E+00	-0.70544E+00	0.48549E-01
0.33147E-15	-0.81650E+00	0.13078E-16	-0.10842E-15
0.97145E-16	-0.11083E-16	0.21436E-18	0.31147E-17
0.99159E-31	-0.17261E-47	0.0	-0.25140E-46
0.47898E-15	-0.67654E-16	0.36699E-01	0.53326E+00
-0.93346E-16	0.13878E-16	0.38519E-33	0.31200E-31
0.70711E+00	0.66787E-16	-0.18350E-01	-0.26663E+00
-0.70711E+00	0.34694E-16	-0.18350E-01	-0.26663E+00
-0.57612E-15	0.78930E-16	-0.36699E-01	-0.53326E+00
-0.49579E-31	0.86304E-48	0.0	0.12570E-46
0.57612E-15	-0.78930E-16	0.36699E-01	0.53326E+00

MULTIPLE EIGENVALUE: -0.20637E-15

CORRESPONDING H(I) MATRIX FOR G1:

0.6190E+00	-.3810E+00	-.4911E-17	-.2381E+00	0.3660E-17	0.1429E+00	-.5029E-16	-.2570E-17	0.9524E-01	-.6028E-17	-.4762E-01	-.4762E-01
-.3810E+00	0.6190E+00	-.4907E-17	-.2381E+00	0.3660E-17	0.1429E+00	-.3859E-16	-.2575E-17	0.9524E-01	-.6028E-17	-.4762E-01	-.4762E-01
-.4911E-17	-.4907E-17	0.1263E-32	0.9818E-17	-.1106E-32	0.4498E-17	-.4341E-34	0.1360E-32	-.1432E-16	0.1784E-32	0.2704E-16	-.1273E-16
-.2381E+00	-.2381E+00	0.9818E-17	0.4762E+00	-.7320E-17	-.2857E+00	0.8889E-16	0.5145E-17	-.1905E+00	0.1206E-16	0.9524E-01	0.9524E-01
0.3660E-17	0.3660E-17	-.1106E-32	-.7320E-17	0.1515E-32	-.1768E-16	-.1290E-32	-.7068E-33	0.2500E-16	-.2079E-32	-.2210E-16	-.2904E-17
0.1429E+00	0.1429E+00	0.4498E-17	-.2857E+00	-.1768E-16	0.5714E+00	-.3390E-16	-.2562E-17	-.2857E+00	0.1980E-16	0.1429E+00	0.1429E+00
-.5029E-16	-.3859E-16	-.4341E-34	0.8889E-16	-.1290E-32	-.3390E-16	0.2477E-31	-.2749E-32	-.5499E-16	0.1055E-32	-.3237E-16	0.8735E-16
-.2570E-17	-.2575E-17	0.1360E-32	0.5145E-17	-.7068E-33	-.2562E-17	-.2749E-32	0.2003E-32	-.2583E-17	0.1473E-32	0.3249E-16	-.2991E-16
0.9524E-01	0.9524E-01	-.1432E-16	-.1905E+00	0.2500E-16	-.2857E+00	-.5498E-16	-.2583E-17	0.4762E+00	-.3186E-16	-.2381E+00	-.2381E+00
-.6028E-17	-.6028E-17	0.1784E-32	0.1206E-16	-.2079E-32	0.1980E-16	0.1055E-32	0.1473E-32	-.3186E-16	0.3011E-32	0.3688E-16	-.5021E-17
-.4762E-01	-.4762E-01	0.2704E-16	0.9524E-01	-.2210E-16	0.1429E+00	-.3237E-16	0.3249E-16	-.2381E+00	0.3688E-16	0.6190E+00	-.3810E+00
-.4762E-01	-.4762E-01	-.1273E-16	0.9524E-01	-.2904E-17	0.1429E+00	0.8735E-16	-.2991E-16	-.2381E+00	-.5021E-17	-.3810E+00	0.6190E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.6667E+00	-.3333E+00	-.3333E+00	-.4525E-17	-.1251E-46	-.2062E-17	0.5666E-17	-.7090E-16	0.8537E-16	0.6239E-17	0.6255E-47	-.6239E-17
-.3333E+00	0.6667E+00	-.3333E+00	-.4525E-17	-.1251E-46	-.3780E-17	0.5666E-17	-.6764E-16	0.8664E-16	0.8810E-17	0.6255E-47	-.8810E-17
-.3333E+00	-.3333E+00	0.6667E+00	0.9049E-17	0.3428E-46	-.2068E-17	-.1133E-16	0.2085E-15	-.2341E-15	-.7139E-17	-.1714E-46	0.7139E-17
-.4525E-17	-.4525E-17	0.9049E-17	0.9570E-32	0.9633E-47	0.1669E-17	-.9222E-32	0.6786E-16	-.6953E-16	-.1669E-17	-.4816E-47	0.1669E-17
-.1251E-46	-.1251E-46	0.3428E-46	0.9633E-47	0.9832E-62	0.3409E-46	-.9256E-47	0.7012E-31	-.7012E-31	-.4372E-46	-.4916E-62	0.4372E-46
-.2062E-17	-.3780E-17	-.2068E-17	0.1669E-17	0.3409E-46	0.2857E+00	-.2900E-31	-.1429E+00	-.1429E+00	-.2857E+00	-.1704E-46	0.2857E+00
0.5666E-17	0.5666E-17	-.1133E-16	-.9222E-32	-.9256E-47	-.2900E-31	0.8906E-32	-.6601E-16	0.6601E-16	0.3822E-31	0.4628E-47	-.3822E-31
-.7090E-16	-.6764E-16	0.2085E-15	0.6786E-16	0.7012E-31	-.1429E+00	-.6601E-16	0.5714E+00	-.4286E+00	0.1429E+00	-.3506E-31	-.1429E+00
0.8537E-16	0.8664E-16	-.2341E-15	-.6953E-16	-.7012E-31	-.1429E+00	0.6601E-16	-.4286E+00	0.5714E+00	0.1429E+00	0.3506E-31	-.1429E+00
0.6239E-17	0.8810E-17	-.7139E-17	-.1669E-17	-.4372E-46	-.2857E+00	0.3822E-31	0.1429E+00	0.1429E+00	0.2857E+00	0.2186E-46	-.2857E+00
0.6255E-47	0.6255E-47	-.1714E-46	-.4816E-47	-.4916E-62	-.1704E-46	0.4628E-47	-.3506E-31	0.3506E-31	0.2186E-46	0.2458E-62	-.2186E-46
-.6239E-17	-.8810E-17	0.7139E-17	0.1669E-17	0.4372E-46	0.2857E+00	-.3822E-31	-.1429E+00	-.1429E+00	-.2857E+00	-.2186E-46	0.2857E+00

EIGENVALUE: 0.78014E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.13504E+00
-0.13504E+00
-0.10535E+00
0.18789E+00
0.25197E+00
0.32293E+00
-0.50217E+00
-0.64369E+00
0.18789E+00
-0.10535E+00
-0.13504E+00
-0.13504E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.19863E+00
-0.19863E+00
-0.19863E+00
-0.15496E+00
0.47501E+00
0.35426E+00
-0.19867E+00
-0.25461E+00
-0.25461E+00
0.17127E+00
-0.34139E+00
-0.43760E+00

EIGENVALUE: 0.17321E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.28868E+00
0.28868E+00
0.50000E+00
0.28868E+00
-0.65082E-15
-0.37334E-15
-0.89614E-15
-0.69434E-15
-0.28868E+00
-0.50000E+00
-0.28868E+00
-0.28868E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
-0.20412E+00
-0.20412E+00
-0.20412E+00
-0.35355E+00
0.35693E-15
0.35355E+00
0.61237E+00
0.35355E+00
0.35355E+00
0.41215E-15
0.35693E-15
0.20607E-15

EIGENVALUE: 0.14924E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
0.27383E+00
0.27383E+00
0.40866E+00
0.62197E-01
-0.31584E+00
-0.21164E+00
-0.38410E+00
-0.25738E+00
0.62197E-01
0.40866E+00
0.27383E+00
0.27383E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.17581E+00
0.17581E+00
0.17581E+00
0.26238E+00
-0.13588E+00
0.26759E-01
0.17581E+00
0.11781E+00
0.11781E+00
-0.49192E+00
-0.59824E+00
-0.40087E+00

EIGENVALUE: 0.22725E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:
-0.11799E+00
-0.11799E+00
-0.26814E+00
-0.37336E+00
-0.58033E+00
-0.25537E+00
-0.31669E+00
-0.13936E+00
-0.37336E+00
-0.26814E+00
-0.11799E+00
-0.11799E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:
0.23374E+00
0.23374E+00
0.23374E+00
0.53117E+00
0.50587E+00
0.32546E+00
0.23374E+00
0.10286E+00
0.10286E+00
0.29296E+00
0.15987E+00
0.70350E-01

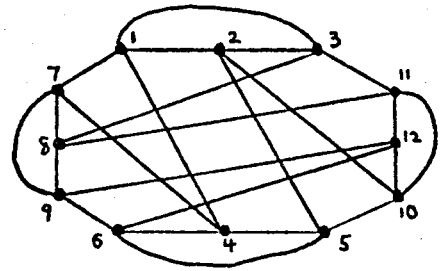
TWO COSPECTRAL GRAPHS

THERE ARE 2 COSPECTRAL GRAPHS ON 12 POINTS IN THIS EXAMPLE

ADJACENCY LISTS

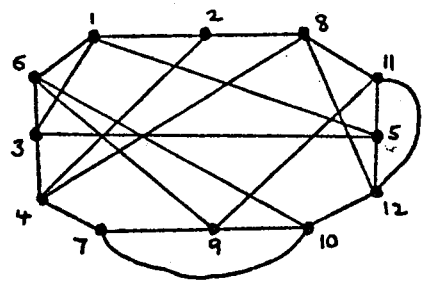
G1

```
 1:  2  3  4  7  0
 2:  3  5 10  0
 3:  8 11  0
 4:  5  6  7  0
 5:  6 10  0
 6:  9 12  0
 7:  8  9  0
 8:  9 11  0
 9: 12  0
10: 11 12  0
11: 12  0
12:  0
```



G2

```
 1:  2  3  5  6  0
 2:  4  7  8  0
 3:  4  5  6  0
 4:  7  8  0
 5: 11 12  0
 6:  9 10  0
 7:  9 10  0
 8: 11 12  0
 9: 10 11  0
10: 12  0
11: 12  0
12:  0
```



Two Non-Isomorphic Graphs

THE EIGENVALUES ARE:

-0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 -0.42293E-15 0.11252E-14 0.15403E-14 0.20000E+01 0.20000E+01
0.20000E+01 0.40000E+01

-0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 0.19975E-14 0.19984E-14 0.26645E-14 0.20000E+01 0.20000E+01
0.20000E+01 0.40000E+01

MULTIPLICITY OF THE EIGENVALUES: 5 3 3 1

EIGENVALUE: -0.20000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.18988E+00	0.44484E+00	-0.10927E+00	0.36952E-01	0.41161E+00
-0.46118E-01	0.10234E+00	-0.53075E-01	0.12571E+00	-0.62084E+00
-0.14377E+00	-0.54718E+00	0.16234E+00	-0.16266E+00	0.20923E+00
-0.66389E-01	0.27178E-03	0.55060E+00	-0.65349E-01	-0.32378E+00
-0.24786E+00	0.13534E-01	-0.10207E+00	0.38527E+00	0.44294E+00
0.31425E+00	-0.13805E-01	-0.44853E+00	-0.31992E+00	-0.11917E+00
-0.12350E+00	-0.44511E+00	-0.44133E+00	0.28397E-01	-0.87836E-01
-0.20177E+00	0.51669E+00	-0.71953E-02	-0.32041E+00	-0.32835E-01
0.32526E+00	-0.71584E-01	0.44853E+00	0.30002E+00	0.12067E+00
0.29397E+00	-0.11538E+00	0.15515E+00	-0.51098E+00	0.17790E+00
0.34553E+00	0.30488E-01	-0.15515E+00	0.49108E+00	-0.17639E+00
-0.63951E+00	0.85390E-01	0.0	0.19903E-01	-0.15959E-02

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.85468E-01	0.86847E-02	0.60916E+00	0.19294E+00	0.31341E-01
-0.97882E-01	-0.30833E+00	-0.55774E+00	-0.11728E-02	0.30669E-01
-0.87196E-01	-0.69941E-01	-0.21268E+00	0.10840E+00	0.58923E+00
-0.96158E-01	-0.22970E+00	0.26410E+00	0.83368E-01	-0.52722E+00
0.34351E+00	0.24598E+00	-0.29240E+00	0.47493E-01	-0.38782E+00
0.12499E-01	0.11492E+00	-0.15550E+00	-0.58060E+00	-0.29476E+00
0.23188E-01	0.35330E+00	0.18956E+00	-0.43102E+00	0.26380E+00
0.35420E+00	0.48437E+00	0.52659E-01	0.15706E+00	0.17074E+00
-0.30660E+00	0.19279E+00	-0.13369E+00	0.50878E+00	-0.93629E-01
0.45427E+00	-0.36136E+00	0.48211E-01	0.27108E+00	0.62577E-01
0.12325E+00	-0.49243E+00	0.18511E+00	-0.31701E+00	0.15564E+00
-0.63762E+00	0.61721E-01	0.32087E-02	-0.79311E-01	-0.56756E-03

MULTIPLE EIGENVALUE: -0.20000E+01

CORRESPONDING H(I) MATRIX FOR G1:

0.4167E+00	-.2083E+00	-.2083E+00	-.2083E+00	0.1667E+00	0.4167E-01	-.2083E+00	0.1667E+00	0.4167E-01	0.4167E-01	0.4167E-01	-.8333E-01
-.2083E+00	0.4167E+00	-.2083E+00	0.1667E+00	-.2083E+00	0.4167E-01	0.4167E-01	0.4167E-01	-.8333E-01	-.2083E+00	0.1667E+00	0.4167E-01
-.2083E+00	-.2083E+00	0.4167E+00	0.4167E-01	0.4167E-01	-.8333E-01	0.1667E+00	-.2083E+00	0.4167E-01	0.1667E+00	-.2083E+00	0.4167E-01
-.2083E+00	0.1667E+00	0.4167E-01	0.4167E+00	-.2083E+00	-.2083E+00	-.2083E+00	0.4167E-01	0.1667E+00	0.4167E-01	-.8333E-01	0.4167E-01
0.1667E+00	-.2083E+00	0.4167E-01	-.2083E+00	0.4167E+00	-.2083E+00	0.4167E-01	-.8333E-01	0.4167E-01	-.2083E+00	0.4167E-01	0.1667E+00
0.4167E-01	0.4167E-01	-.8333E-01	-.2083E+00	-.2083E+00	0.4167E+00	0.1667E+00	0.4167E-01	-.2083E+00	0.1667E+00	0.4167E-01	-.2083E+00
-.2083E+00	0.4167E-01	0.1667E+00	-.2083E+00	0.4167E-01	0.1667E+00	0.4167E+00	-.2083E+00	-.2083E+00	-.8333E-01	0.4167E-01	0.4167E-01
0.1667E+00	0.4167E-01	-.2083E+00	0.4167E-01	-.8333E-01	0.4167E-01	-.2083E+00	0.4167E+00	-.2083E+00	0.4167E-01	-.2083E+00	0.1667E+00
0.4167E-01	-.8333E-01	0.4167E-01	0.1667E+00	0.4167E-01	-.2083E+00	-.2083E+00	-.2083E+00	0.4167E+00	0.4167E-01	0.1667E+00	-.2083E+00
0.4167E-01	-.2083E+00	0.1667E+00	0.4167E-01	-.2083E+00	0.1667E+00	-.8333E-01	0.4167E-01	0.4167E-01	0.4167E+00	-.2083E+00	-.2083E+00
0.4167E-01	0.1667E+00	-.2083E+00	-.8333E-01	0.4167E-01	0.4167E-01	0.4167E-01	-.2083E+00	0.1667E+00	-.2083E+00	0.4167E+00	-.2083E+00
-.8333E-01	0.4167E-01	0.4167E-01	0.4167E-01	0.1667E+00	-.2083E+00	0.4167E-01	0.1667E+00	-.2083E+00	-.2083E+00	-.2083E+00	0.4167E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.4167E+00	-.3333E+00	-.8333E-01	0.1667E+00	-.2083E+00	-.2083E+00	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01
-.3333E+00	0.4167E+00	0.1667E+00	-.8333E-01	0.4167E-01	0.4167E-01	-.2083E+00	-.2083E+00	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01
-.8333E-01	0.1667E+00	0.4167E+00	-.3333E+00	-.2083E+00	-.2083E+00	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01
0.1667E+00	-.8333E-01	-.3333E+00	0.4167E+00	0.4167E-01	0.4167E-01	-.2083E+00	-.2083E+00	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01
-.2083E+00	0.4167E-01	-.2083E+00	0.4167E-01	0.4167E+00	0.1667E+00	-.8333E-01	0.1667E+00	0.4167E-01	0.4167E-01	-.2083E+00	-.2083E+00
-.2083E+00	0.4167E-01	-.2083E+00	0.4167E-01	0.1667E+00	0.4167E+00	0.1667E+00	-.8333E-01	-.2083E+00	-.2083E+00	0.4167E-01	0.4167E-01
0.4167E-01	-.2083E+00	0.4167E-01	-.2083E+00	-.8333E-01	0.1667E+00	0.4167E+00	0.1667E+00	-.2083E+00	-.2083E+00	0.4167E-01	0.4167E-01
0.4167E-01	-.2083E+00	0.4167E-01	-.2083E+00	0.1667E+00	-.8333E-01	0.1667E+00	0.4167E+00	0.4167E-01	-.2083E+00	-.2083E+00	
0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	-.2083E+00	-.2083E+00	0.4167E-01	-.8333E-01	0.4167E+00	0.1667E+00	-.3333E+00
0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	-.2083E+00	0.4167E-01	0.4167E-01	-.2083E+00	-.3333E+00	0.1667E+00	0.4167E+00	-.8333E-01
0.4167E-01	0.4167E-01	0.4167E-01	0.4167E-01	-.2083E+00	0.4167E-01	0.4167E-01	-.2083E+00	0.1667E+00	-.3333E+00	-.8333E-01	0.4167E+00

EIGENVALUE: -0.42293E-15

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.18700E-01	-0.71119E-02	-0.49960E+00
0.49072E+00	-0.94348E-01	-0.17024E-01
0.94030E-01	0.49097E+00	-0.10508E-01
-0.49072E+00	0.94348E-01	0.17024E-01
0.18700E-01	0.71119E-02	0.49960E+00
0.94030E-01	0.49097E+00	-0.10508E-01
-0.94030E-01	-0.49097E+00	0.10508E-01
0.18700E-01	0.71119E-02	0.49960E+00
0.49072E+00	-0.94348E-01	-0.17024E-01
-0.94030E-01	-0.49097E+00	0.10508E-01
-0.49072E+00	0.94348E-01	0.17024E-01
-0.18700E-01	-0.71119E-02	-0.49960E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.26021E-16	0.65905E-16	-0.50000E+00
0.12403E-15	0.64647E-16	-0.50000E+00
-0.52042E-17	0.65905E-16	0.50000E+00
-0.37297E-16	0.64647E-16	0.50000E+00
0.50000E+00	-0.71133E-16	0.59742E-16
-0.50000E+00	0.18620E-16	-0.47867E-16
0.50000E+00	-0.65037E-16	0.48146E-16
-0.50000E+00	0.12605E-16	-0.40199E-16
-0.70536E-16	-0.50000E+00	-0.13143E-19
-0.67735E-16	0.50000E+00	-0.19408E-16
-0.10491E-15	-0.50000E+00	0.16561E-17
0.41747E-16	0.50000E+00	0.32671E-19

MULTIPLE EIGENVALUE: -0.42293E-15

CORRESPONDING H(I) MATRIX FOR G1:

0.2500E+00	-.5786E-16	0.1078E-15	0.1076E-15	-.2500E+00	0.1208E-17	-.4999E-16	-.2500E+00	-.1535E-15	-.4463E-17	0.6756E-16	0.2500E+00
-.5786E-16	0.2500E+00	0.9909E-16	-.2500E+00	-.2445E-16	0.2255E-15	-.1894E-15	-.8879E-16	0.2500E+00	-.1260E-15	-.2500E+00	0.8964E-16
0.1078E-15	0.9909E-16	0.2500E+00	-.7865E-16	-.4873E-16	0.2500E+00	-.2500E+00	-.1429E-15	-.9581E-18	-.2500E+00	0.1559E-15	0.7279E-16
0.1076E-15	-.2500E+00	-.7865E-16	0.2500E+00	-.2525E-16	-.2050E-15	0.1690E-15	0.3909E-16	-.2500E+00	0.1056E-15	0.2500E+00	-.3995E-16
-.2500E+00	-.2445E-16	-.4873E-16	-.2525E-16	0.2500E+00	0.5784E-16	0.2500E+00	-.2500E+00	0.7115E-16	-.5453E-15	0.1475E-16	-.2500E+00
0.1208E-17	0.2255E-15	0.2500E+00	-.2050E-15	0.5784E-16	0.2500E+00	-.2500E+00	-.3635E-16	0.1254E-15	-.2500E+00	0.2954E-16	-.3379E-16
-.4999E-16	-.1894E-15	-.2500E+00	0.1690E-15	-.9057E-17	-.2500E+00	0.2500E+00	0.8513E-16	-.8935E-16	0.2500E+00	-.6560E-16	-.1499E-16
-.2500E+00	-.8879E-16	-.1429E-15	0.3909E-16	0.2500E+00	-.3635E-16	0.8513E-16	0.2500E+00	0.6813E-17	0.3961E-15	0.7909E-16	-.2500E+00
-.1535E-15	0.2500E+00	-.9581E-18	-.2500E+00	0.7115E-16	0.1254E-15	-.8935E-16	0.6813E-17	0.2500E+00	-.2599E-16	-.2500E+00	-.5956E-17
-.4468E-17	-.1260E-15	-.2500E+00	0.1056E-15	-.5458E-16	-.2500E+00	0.2500E+00	0.3961E-16	-.2599E-16	0.2500E+00	-.1290E-15	0.3053E-16
0.6756E-16	-.2500E+00	0.1559E-15	0.2500E+00	0.1475E-16	0.2954E-15	-.6560E-16	0.7909E-16	-.2500E+00	-.1290E-15	0.2500E+00	-.7995E-16
0.2500E+00	0.8964E-16	0.7279E-16	-.3995E-16	-.2500E+00	-.3379E-16	-.1499E-16	-.2500E+00	-.5956E-17	0.3053E-16	-.7995E-16	0.2500E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.2500E+00	0.2500E+00	-.2500E+00	-.2500E+00	-.4288E-16	0.3694E-16	-.3708E-16	0.3311E-16	-.3295E-16	0.4266E-16	-.3378E-16	0.3294E-16
0.2500E+00	0.2500E+00	-.2500E+00	-.2500E+00	0.3215E-16	-.3808E-16	0.3794E-16	-.4192E-16	0.4232E-16	-.3315E-16	0.3231E-16	
-.2500E+00	-.2500E+00	0.2500E+00	0.2500E+00	0.2727E-16	-.2133E-16	0.2147E-16	-.1750E-16	-.3296E-16	0.2325E-16	-.3212E-16	0.3297E-16
-.2500E+00	-.2500E+00	0.2500E+00	0.2500E+00	0.1122E-16	-.5285E-17	0.5425E-17	-.1451E-17	-.3233E-16	0.2262E-16	-.3150E-16	0.3234E-16
-.4288E-16	0.3215E-16	0.2727E-16	0.1122E-16	0.2500E+00	-.2500E+00	0.2500E+00	-.2500E+00	0.3014E-18	-.6944E-16	-.1688E-16	-.1470E-16
0.3694E-16	-.3808E-16	-.2133E-16	-.5285E-17	-.2500E+00	0.2500E+00	-.2500E+00	0.2500E+00	0.2596E-16	0.4318E-16	0.4314E-16	-.1156E-16
-.3708E-16	0.3794E-16	0.2147E-16	0.5425E-17	0.2500E+00	-.2500E+00	0.2500E+00	-.2500E+00	-.2749E-17	-.6639E-16	-.1993E-16	-.1165E-16
0.3311E-16	-.4192E-16	-.1750E-16	-.1451E-17	-.2500E+00	0.2500E+00	-.2500E+00	0.2500E+00	0.2897E-16	0.4017E-16	0.4615E-16	-.1457E-16
-.3295E-16	-.3232E-16	-.3296E-16	-.3233E-16	0.3014E-18	0.2596E-16	-.2749E-17	0.2897E-16	0.2500E+00	-.2500E+00	0.2500E+00	-.2500E+00
0.4266E-16	0.4203E-16	0.2325E-16	0.2262E-16	-.6944E-16	0.4318E-16	-.6639E-16	0.4017E-16	-.2500E+00	0.2500E+00	-.2500E+00	0.2500E+00
-.3378E-16	-.3315E-16	-.3212E-16	-.3150E-16	-.1688E-16	0.4314E-16	-.1993E-16	0.4615E-16	0.2500E+00	-.2500E+00	0.2500E+00	-.2500E+00
0.3294E-16	0.3231E-16	0.3297E-16	0.3234E-16	-.1470E-16	-.1156E-16	-.1165E-16	-.1457E-16	-.2500E+00	0.2500E+00	-.2500E+00	0.2500E+00

EIGENVALUE: 0.20000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.23592E-15	0.50000E+00	0.12490E-15
0.32464E+00	0.25000E+00	-0.28654E+00
0.37837E+00	0.25000E+00	0.21056E+00
-0.37837E+00	0.25000E+00	-0.21056E+00
-0.53727E-01	0.30097E-15	-0.49711E+00
-0.37837E+00	-0.25000E+00	-0.21056E+00
-0.32464E+00	0.25000E+00	0.28654E+00
0.53727E-01	0.91940E-16	0.49711E+00
-0.32464E+00	-0.25000E+00	0.28654E+00
0.32464E+00	-0.25000E+00	-0.28654E+00
0.37837E+00	-0.25000E+00	0.21056E+00
-0.97145E-16	-0.50000E+00	-0.83267E-16

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.21668E+00	-0.37490E+00	0.25000E+00
0.42569E+00	-0.79318E-01	0.25000E+00
-0.21668E+00	-0.37490E+00	0.25000E+00
0.42569E+00	-0.79318E-01	0.25000E+00
-0.21668E+00	-0.37490E+00	-0.25000E+00
-0.42569E+00	0.79318E-01	0.25000E+00
0.21668E+00	0.37490E+00	0.25000E+00
0.42569E+00	-0.79318E-01	-0.25000E+00
-0.20901E+00	0.45422E+00	0.25153E-16
-0.20901E+00	0.45422E+00	-0.39812E-15
-0.23574E-15	-0.56455E-16	-0.50000E+00
0.38118E-16	-0.80310E-16	-0.50000E+00

MULTIPLE EIGENVALUE: 0.20000E+01

CORRESPONDING H(I) MATRIX FOR G1:

0.2500E+00	0.1250E+00	0.1250E+00	0.1250E+00	0.7572E-16	-.1250E+00	0.1250E+00	0.1207E-15	-.1250E+00	-.1250E+00	-.1250E+00	-.2500E+00
0.1250E+00	0.2500E+00	0.1250E+00	-.2978E-16	0.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.2500E+00	0.1250E+00	0.6857E-16	-.1250E+00
0.1250E+00	0.1250E+00	0.2500E+00	-.1250E+00	-.1250E+00	-.2500E+00	0.1633E-15	0.1250E+00	-.1250E+00	-.5315E-16	0.1250E+00	-.1250E+00
0.1250E+00	-.2978E-16	-.1250E+00	0.2500E+00	0.1250E+00	0.1250E+00	0.1250E+00	-.1250E+00	-.3265E-16	-.1250E+00	-.2500E+00	-.1250E+00
0.7572E-16	0.1250E+00	-.1250E+00	0.1250E+00	0.2500E+00	0.1250E+00	-.1250E+00	-.2500E+00	-.1250E+00	0.1250E+00	-.1250E+00	-.1039E-15
-.1250E+00	-.1250E+00	-.2500E+00	0.1250E+00	0.1250E+00	0.2500E+00	-.2550E-16	-.1250E+00	0.1250E+00	-.7969E-16	-.1250E+00	0.1250E+00
0.1250E+00	-.1250E+00	0.1633E-15	0.1250E+00	-.1250E+00	-.2550E-16	0.2500E+00	0.1250E+00	0.1250E+00	-.2500E+00	-.1250E+00	-.1250E+00
0.1207E-15	-.1250E+00	0.1250E+00	-.1250E+00	-.2500E+00	-.1250E+00	0.1250E+00	0.2500E+00	0.1250E+00	-.1250E+00	0.1250E+00	-.9258E-16
-.1250E+00	-.2500E+00	-.1250E+00	-.3265E-16	-.1250E+00	0.1250E+00	0.1250E+00	0.1250E+00	0.2500E+00	-.1250E+00	-.6142E-17	0.1250E+00
-.1250E+00	0.1250E+00	-.5815E-16	-.1250E+00	0.1250E+00	-.7969E-16	-.2500E+00	-.1250E+00	-.1250E+00	0.2500E+00	0.1250E+00	0.1250E+00
-.1250E+00	0.6857E-16	0.1250E+00	-.2500E+00	-.1250E+00	-.1250E+00	-.1250E+00	0.1250E+00	-.6142E-17	0.1250E+00	0.2500E+00	0.1250E+00
-.2500E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1039E-15	0.1250E+00	-.1250E+00	-.9258E-16	0.1250E+00	0.1250E+00	0.1250E+00	0.2500E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.2500E+00	-.3083E-16	0.2500E+00	-.3083E-16	0.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00
-.3083E-16	0.2500E+00	-.3083E-16	0.2500E+00	-.1250E+00	-.1250E+00	0.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00
0.2500E+00	-.3083E-16	0.2500E+00	-.3083E-16	0.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00
-.3083E-16	0.2500E+00	-.3083E-16	0.2500E+00	-.1250E+00	-.1250E+00	0.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00
0.1250E+00	-.1250E+00	0.1250E+00	-.1250E+00	0.2500E+00	-.5369E-16	-.2500E+00	-.6806E-16	-.1250E+00	-.1250E+00	0.1250E+00	0.1250E+00
0.1250E+00	-.1250E+00	0.1250E+00	-.1250E+00	-.5369E-16	0.2500E+00	0.9214E-16	-.2500E+00	0.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00
-.1250E+00	0.1250E+00	-.1250E+00	0.1250E+00	-.2500E+00	0.9214E-16	0.2500E+00	0.2961E-16	0.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00
-.1250E+00	0.1250E+00	-.1250E+00	0.1250E+00	-.6806E-16	-.2500E+00	0.2961E-16	0.2500E+00	-.1250E+00	-.1250E+00	0.1250E+00	0.1250E+00
-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	0.1250E+00	0.1250E+00	-.1250E+00	0.2500E+00	0.2500E+00	0.1105E-16	-.5702E-16
-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	0.1250E+00	0.1250E+00	-.1250E+00	0.2500E+00	0.2500E+00	0.2227E-15	0.1546E-15
-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00	0.1250E+00	0.1105E-16	0.2227E-15	0.2500E+00	0.2500E+00
-.1250E+00	-.1250E+00	-.1250E+00	-.1250E+00	0.1250E+00	-.1250E+00	-.1250E+00	0.1250E+00	-.5702E-16	0.1546E-15	0.2500E+00	0.2500E+00

EIGENVALUE: 0.40000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00
-0.28868E+00

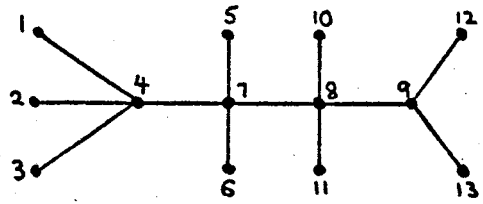
TWO COSPECTRAL GRAPHS

THERE ARE 2 COSPECTRAL GRAPHS ON 13 POINTS IN THIS EXAMPLE

ADJACENCY LISTS

G1

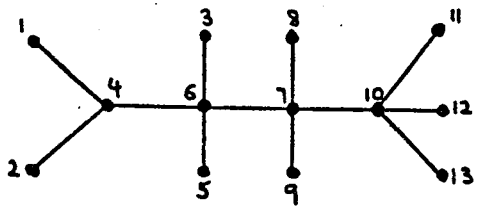
```
 1:  4  0
 2:  4  0
 3:  4  0
 4:  7  0
 5:  7  0
 6:  7  0
 7:  8  0
 8:  9 10 11  0
 9: 12 13  0
10:  0
11:  0
12:  0
13:  0
```



Two Isomorphic Trees

G2

```
 1:  4  0
 2:  4  0
 3:  6  0
 4:  6  0
 5:  6  0
 6:  7  0
 7:  8  9 10  0
 8:  0
 9:  0
10: 11 12 13  0
11:  0
12:  0
13:  0
```



THE EIGENVALUES ARE:

-0.24915E+01 -0.18853E+01 -0.12348E+01 -0.84465E+00 -0.71148E-16 -0.30239E-49 0.77037E-33 0.28911E-16 0.14877E-15 0.84465E+00
0.12348E+01 0.18853E+01 0.24915E+01

-0.24915E+01 -0.18853E+01 -0.12348E+01 -0.84465E+00 -0.15754E-15 -0.48002E-16 -0.38519E-33 0.69454E-17 0.73212E-16 0.84465E+00
0.12348E+01 0.18853E+01 0.24915E+01

MULTIPLICITY OF THE EIGENVALUES: 1 1 1 1 5 1 1 1 1

EIGENVALUE: -0.24915E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.16203E+00
-0.16203E+00
-0.16203E+00
0.40370E+00
0.20860E+00
0.20860E+00
-0.51971E+00
0.47398E+00
-0.28065E+00
-0.19023E+00
-0.19023E+00
0.11265E+00
0.11265E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.11265E+00
-0.11265E+00
0.19023E+00
0.28065E+00
0.19023E+00
-0.47398E+00
0.51971E+00
-0.20860E+00
-0.20860E+00
-0.40370E+00
0.16203E+00
0.16203E+00
0.16203E+00

EIGENVALUE: -0.12343E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.23891E+00
-0.23891E+00
-0.23891E+00
0.29500E+00
-0.28546E+00
-0.28546E+00
0.35248E+00
-0.15930E+00
-0.41381E+00
0.12902E+00
0.12902E+00
0.33513E+00
0.33513E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.33513E+00
-0.33513E+00
-0.12902E+00
0.41381E+00
-0.12902E+00
0.15930E+00
-0.35248E+00
0.28546E+00
0.28546E+00
-0.29500E+00
0.23891E+00
0.23891E+00
0.23891E+00

EIGENVALUE: -0.18853E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.25898E+00
0.25898E+00
0.25898E+00
-0.48826E+00
-0.76170E-01
-0.76170E-01
0.14361E+00
0.36985E+00
-0.44856E+00
-0.19617E+00
-0.19617E+00
0.23792E+00
0.23792E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.23792E+00
0.23792E+00
-0.19617E+00
-0.44856E+00
-0.19617E+00
0.36985E+00
0.14361E+00
-0.76170E-01
-0.76170E-01
-0.48826E+00
0.25898E+00
0.25898E+00
0.25898E+00

EIGENVALUE: -0.84465E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.12753E+00
-0.12753E+00
-0.12753E+00
0.10772E+00
-0.34525E+00
-0.34525E+00
0.29162E+00
0.33647E+00
0.22090E+00
-0.39835E+00
-0.39835E+00
-0.26152E+00
-0.26152E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.26152E+00
0.26152E+00
0.39835E+00
-0.22090E+00
0.39835E+00
-0.33647E+00
-0.29162E+00
0.34525E+00
0.34525E+00
-0.10772E+00
0.12753E+00
0.12753E+00
0.12753E+00

EIGENVALUE: -0.71143E-16

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.17688E-15	0.70711E+00	0.40825E+00	0.15223E-15	0.15207E-15
0.17275E-15	-0.70711E+00	0.40825E+00	0.18334E-15	0.14761E-15
0.18544E-15	0.41633E-16	-0.81650E+00	0.16245E-15	0.13776E-15
0.49670E-16	-0.77037E-33	-0.23527E-16	-0.82042E-16	-0.19195E-15
0.10778E+00	0.12004E-16	-0.14354E-17	0.48831E+00	-0.49994E+00
-0.10778E+00	-0.12004E-16	0.33892E-16	-0.48831E+00	0.49994E+00
0.70491E-17	-0.13482E-32	-0.78104E-17	-0.74192E-16	0.11313E-16
-0.67436E-16	0.34667E-32	-0.29768E-17	0.16364E-15	0.14363E-15
0.41147E-16	0.0	0.0	0.10012E-16	0.31772E-16
0.39233E+00	0.57078E-17	-0.29334E-16	0.37634E+00	0.45216E+00
-0.39233E+00	-0.57078E-17	0.37144E-16	-0.37634E+00	-0.45216E+00
0.57833E+00	-0.86312E-17	0.11608E-16	-0.34630E+00	-0.21357E+00
-0.57833E+00	0.86312E-17	-0.86312E-17	0.34630E+00	0.21357E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.21141E-15	0.14957E-15	0.70711E+00	-0.30766E-15	0.86270E-16
-0.21125E-15	0.14772E-15	-0.70711E+00	0.33096E-15	0.86027E-16
0.45592E-02	0.62179E-02	0.16181E-15	0.70704E+00	-0.58622E-02
0.56491E-16	0.14671E-16	0.17771E-20	-0.10324E-16	0.14872E-16
-0.45592E-02	-0.62179E-02	-0.16182E-15	-0.70704E+00	0.58622E-02
0.13802E-15	-0.44662E-16	-0.46535E-19	0.56240E-17	-0.72630E-16
-0.65547E-16	-0.12767E-15	0.89357E-21	-0.14562E-16	-0.29690E-16
0.19130E+00	-0.67804E+00	0.53072E-18	0.52293E-02	0.60307E-01
-0.19130E+00	0.67804E+00	-0.48418E-18	-0.52293E-02	-0.60307E-01
0.63906E-16	0.84253E-17	0.15053E-51	-0.10645E-19	-0.50261E-16
0.78589E+00	0.21858E+00	-0.11863E-17	-0.72776E-02	-0.34692E-01
-0.38025E+00	-0.43066E-01	0.15762E-17	0.88104E-02	0.72121E+00
-0.40565E+00	-0.17552E+00	-0.39079E-18	-0.15327E-02	-0.68652E+00

MULTIPLE EIGENVALUE: -0.71148E-16

CORRESPONDING H(I) MATRIX FOR G1:

0.6667E+00	-.3333E+00	-.3333E+00	-.9605E-17	0.2528E-16	-.1203E-16	-.3189E-17	-.1215E-17	0.1363E-31	0.1875E-15	-.1843E-15	0.1573E-16	-.1452E-16
-.3333E+00	0.6667E+00	-.3333E+00	-.9605E-17	0.2528E-16	-.1203E-16	-.3189E-17	-.1215E-17	0.1363E-31	0.1875E-15	-.1843E-15	0.1573E-16	-.1452E-16
-.3333E+00	-.3333E+00	0.6667E+00	0.1921E-16	0.3161E-16	-.5811E-16	0.6377E-17	0.2431E-17	0.1363E-31	0.2201E-15	-.2265E-15	0.1209E-16	-.1452E-16
-.9605E-17	-.9605E-17	0.1921E-16	0.4660E-31	0.6125E-16	-.6125E-16	0.4449E-32	-.4427E-31	-.4876E-32	-.9818E-16	0.9813E-16	0.9813E-16	-.9813E-16
0.2528E-16	0.2528E-16	0.3161E-16	0.6125E-16	0.5000E+00	-.5000E+00	-.4112E-16	0.8293E-18	-.6561E-17	-.1368E-15	0.2069E-15	-.1070E-15	-.1058E-15
-.1203E-16	-.1203E-15	-.5811E-16	-.6125E-16	-.5000E+00	0.5000E+00	0.4112E-16	-.8293E-18	0.6561E-17	-.1300E-15	0.5985E-16	0.6840E-17	-.8131E-17
-.3189E-17	-.3189E-17	0.6377E-17	0.4449E-32	-.4112E-16	0.4112E-16	0.5743E-32	-.1097E-31	-.9331E-34	-.2004E-15	0.2004E-16	0.2735E-16	-.2735E-16
-.1215E-17	-.1215E-17	0.2431E-17	-.4427E-31	0.8293E-18	-.8293E-18	-.1097E-31	0.5196E-31	0.3427E-32	0.1001E-15	-.1001E-15	-.1263E-15	0.1263E-15
0.1363E-31	0.1363E-31	0.1363E-31	-.4876E-32	-.6561E-17	0.6561E-17	-.9331E-34	0.3427E-32	0.2803E-32	0.3428E-16	-.3428E-16	0.1354E-16	-.1354E-16
0.1875E-15	0.1875E-15	0.2201E-15	-.9818E-16	-.1368E-15	-.1300E-15	-.2004E-16	0.1001E-15	0.3428E-16	0.5000E+00	-.5000E+00	0.6364E-16	-.1556E-15
-.1843E-15	-.1843E-15	-.2265E-15	0.9818E-16	0.2069E-15	0.5985E-16	0.2004E-16	-.1001E-15	-.3428E-16	-.5000E+00	0.5000E+00	-.1467E-15	0.2386E-15
0.1573E-16	0.1573E-16	0.1209E-16	0.9813E-16	0.1070E-15	0.6840E-17	0.2735E-16	-.1263E-15	0.1354E-16	0.6364E-16	-.1467E-15	0.5000E+00	-.5000E+00
-.1452E-16	-.1452E-15	-.1452E-16	-.9813E-16	-.1058E-15	-.8131E-17	-.2735E-16	0.1263E-15	-.1354E-16	-.1556E-15	0.2386E-15	-.5000E+00	0.5000E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.5000E+00	-.5000E+00	-.1035E-15	0.1257E-20	0.1036E-15	-.3291E-19	0.6319E-21	-.1379E-15	0.1379E-15	-.1658E-31	-.1350E-15	0.1346E-15	0.4754E-18
-.5000E+00	0.5000E+00	0.1190E-15	-.1257E-20	-.1190E-15	0.3291E-19	-.6319E-21	0.1340E-15	-.1340E-15	0.1658E-31	-.1333E-15	0.1378E-15	-.4754E-18
-.1036E-15	0.1190E-15	0.5000E+00	-.7035E-17	-.5000E+00	0.4754E-17	-.1121E-16	-.4456E-16	0.3494E-16	0.6309E-18	-.4095E-16	0.6142E-16	-.9512E-17
0.1257E-20	-.1257E-20	-.7035E-17	0.3722E-32	0.7035E-17	0.6033E-32	-.5855E-32	0.1678E-17	-.1678E-17	0.3006E-32	0.4718E-16	-.1177E-16	-.3541E-16
0.1036E-15	-.1190E-15	-.5000E+00	0.7035E-17	0.5000E+00	-.4754E-17	0.1121E-16	0.9060E-16	-.8098E-16	-.6309E-18	-.2333E-17	-.1879E-17	-.6754E-17
-.3291E-19	0.3291E-19	0.4754E-17	0.6033E-32	-.4754E-17	0.2635E-31	-.1270E-32	0.5234E-16	-.5234E-16	0.1209E-31	0.1912E-15	-.1329E-15	0.1704E-17
0.6319E-21	-.6319E-21	-.1121E-16	-.5855E-32	0.1121E-16	-.1270E-32	0.2169E-31	0.7216E-16	-.7216E-16	-.3772E-32	-.7828E-16	0.8881E-17	0.6940E-16
-.1379E-15	-.1340E-15	-.4456E-16	0.1678E-17	0.9060E-16	-.5234E-16	-.7216E-16	0.5000E+00	-.5000E+00	0.3481E-17	-.2304E-15	0.1377E-15	-.4170E-16
0.1379E-15	0.1340E-15	0.3494E-16	-.1678E-17	-.8098E-16	0.5234E-16	-.7216E-16	-.5000E+00	0.5000E+00	-.3481E-17	-.8754E-16	0.2108E-15	-.7217E-16
-.1658E-31	-.1658E-31	0.6309E-18	0.3006E-32	-.6309E-18	0.1209E-31	-.3772E-32	0.3481E-17	-.3481E-17	0.6681E-32	0.5381E-16	-.6091E-16	0.7104E-17
-.1350E-15	-.1383E-15	-.4095E-16	0.4718E-16	-.2333E-17	0.1012E-15	-.7823E-16	-.2304E-15	-.8754E-16	0.5381E-16	0.6667E+00	-.3333E+00	-.3333E+00
0.1346E-15	0.1378E-15	0.6142E-16	-.1177E-16	-.1879E-17	-.1029E-15	0.8881E-16	0.1377E-15	0.2108E-15	-.6091E-16	-.3333E+00	0.6667E+00	-.3333E+00
0.4754E-18	0.4754E-18	-.9512E-17	-.3541E-16	-.6754E-17	0.1704E-17	0.6940E-16	-.4170E-16	-.7217E-16	0.7104E-17	-.3333E+00	-.3333E+00	0.6667E+00

EIGENVALUE: 0.84465E+00

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.12753E+00
-0.12753E+00
-0.12753E+00
-0.10772E+00
0.34525E+00
0.34525E+00
0.29162E+00
-0.33647E+00
0.22090E+00
-0.39835E+00
-0.39835E+00
0.26152E+00
0.26152E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.26152E+00
-0.26152E+00
0.39835E+00
-0.22090E+00
0.39835E+00
0.33647E+00
-0.29162E+00
-0.34525E+00
-0.34525E+00
0.10772E+00
0.12753E+00
0.12753E+00
0.12753E+00

EIGENVALUE: 0.18853E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.25898E+00
0.25898E+00
0.25898E+00
0.48826E+00
0.76170E-01
0.76170E-01
0.14361E+00
-0.36985E+00
-0.44856E+00
-0.19617E+00
-0.19617E+00
-0.23792E+00
-0.23792E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.23792E+00
0.23792E+00
0.19617E+00
0.44856E+00
0.19617E+00
0.36985E+00
-0.14361E+00
-0.76170E-01
-0.76170E-01
-0.48826E+00
-0.25898E+00
-0.25898E+00
-0.25898E+00

EIGENVALUE: 0.12349E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.23891E+00
0.23891E+00
0.23891E+00
0.29500E+00
-0.28546E+00
-0.28546E+00
-0.35248E+00
-0.15930E+00
0.41381E+00
-0.12902E+00
-0.12902E+00
0.33513E+00
0.33513E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.33513E+00
-0.33513E+00
0.12902E+00
-0.41381E+00
0.12902E+00
0.15930E+00
0.35248E+00
0.28546E+00
0.28546E+00
-0.29500E+00
-0.23891E+00
-0.23891E+00
-0.23891E+00

EIGENVALUE: 0.24915E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.16203E+00
0.16203E+00
0.16203E+00
0.40370E+00
0.20860E+00
0.20860E+00
0.51971E+00
0.47394E+00
0.28065E+00
0.19023E+00
0.19023E+00
0.11265E+00
0.11265E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.11265E+00
0.11265E+00
0.19023E+00
0.28065E+00
0.19023E+00
0.47394E+00
0.51971E+00
0.20860E+00
0.20860E+00
0.40370E+00
0.16203E+00
0.16203E+00
0.16203E+00

TWO COSPECTRAL GRAPHS

THERE ARE 2 COSPECTRAL GRAPHS ON 16 POINTS IN THIS EXAMPLE

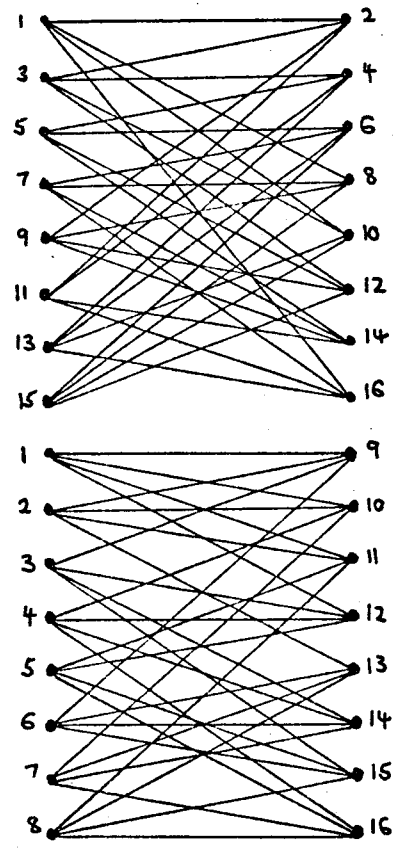
ADJACENCY LISTS

G1

```
 1:  2  8 10 16  0
 2:  3  9 11  0
 3:  4 10 12  0
 4:  5 11 13  0
 5:  6 12 14  0
 6:  7 13 15  0
 7:  8 14 16  0
 8:  9 15  0
 9: 12 14  0
10: 13 15  0
11: 14 16  0
12: 15  0
13: 16  0
14:  0
15:  0
16:  0
```

G2

```
 1:  9 10 11 12  0
 2:  9 10 11 13  0
 3:  9 12 14 15  0
 4: 10 12 14 16  0
 5: 11 12 15 16  0
 6:  9 13 14 15  0
 7: 10 13 14 16  0
 8: 11 13 15 16  0
 9:  0
10:  0
11:  0
12:  0
13:  0
14:  0
15:  0
16:  0
```



Two Non-Isomorphic Graphs

THE EIGENVALUES ARE:

```
-0.40000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 -0.13551E-15 -0.52313E-16 -0.15499E-16  0.15499E-16  0.52313E-16
 0.13551E-15  0.20000E+01  0.20000E+01  0.20000E+01  0.20000E+01  0.40000E+01

-0.40000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 -0.20000E+01 -0.79094E-15 -0.18139E-15 -0.40305E-16  0.11058E-15  0.12063E-15
 0.24978E-15  0.20000E+01  0.20000E+01  0.20000E+01  0.20000E+01  0.40000E+01
```

MULTIPLICITY OF THE EIGENVALUES: 1 4 6 4 1

EIGENVALUE: -0.40000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.25000E+00
0.25000E+00
-0.25000E+00
0.25000E+00
-0.25000E+00
0.25000E+00
-0.25000E+00
0.25000E+00
-0.25000E+00
0.25000E+00
-0.25000E+00
0.25000E+00
-0.25000E+00
0.25000E+00
-0.25000E+00
0.25000E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
0.25000E+00
0.25000E+00
0.25000E+00
0.25000E+00
0.25000E+00
0.25000E+00
0.25000E+00
0.25000E+00

EIGENVALUE: -0.20000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.25000E+00	-0.39165E+00	-0.14740E+00	-0.11128E+00
-0.15025E-16	0.41390E+00	-0.27829E+00	0.35189E-01
-0.25000E+00	-0.13464E+00	0.20370E+00	-0.35760E+00
0.57183E-16	-0.25701E+00	-0.35110E+00	0.24632E+00
-0.25000E+00	0.39165E+00	0.14740E+00	0.11128E+00
-0.18495E-16	-0.41390E+00	0.27829E+00	-0.35189E-01
0.25000E+00	0.13464E+00	-0.20370E+00	0.35760E+00
-0.62676E-16	0.25701E+00	0.35110E+00	-0.24632E+00
-0.25000E+00	-0.27926E+00	0.74593E-01	0.32241E+00
0.13525E-15	0.11239E+00	0.22199E+00	0.43369E+00
0.25000E+00	-0.22251E-01	0.42569E+00	0.76090E-01
0.50000E+00	-0.14510E-16	-0.41706E-16	-0.20209E-15
0.25000E+00	0.27926E+00	-0.74593E-01	-0.32241E+00
-0.18747E-15	-0.11239E+00	-0.22199E+00	-0.43369E+00
-0.25000E+00	0.22251E-01	-0.42569E+00	-0.76090E-01
-0.50000E+00	0.24533E-17	0.42932E-17	0.13639E-15

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.25000E+00	-0.37942E+00	0.19489E+00	-0.74562E-01
-0.25000E+00	0.49411E-01	0.42902E+00	0.31693E-01
-0.25000E+00	-0.49411E-01	-0.42902E+00	-0.31693E-01
0.25000E+00	-0.13808E+00	-0.10038E+00	-0.39794E+00
0.25000E+00	-0.29074E+00	-0.13374E+00	0.29168E+00
-0.25000E+00	0.37942E+00	-0.19489E+00	0.74562E-01
0.25000E+00	0.29074E+00	0.13374E+00	-0.29168E+00
0.25000E+00	0.13808E+00	0.10038E+00	0.39794E+00
0.50000E+00	-0.33408E-15	-0.13818E-15	-0.18004E-15
0.50678E-16	0.88674E-01	-0.32863E+00	0.36625E+00
-0.66958E-16	0.24133E+00	-0.29527E+00	-0.32338E+00
0.99116E-17	0.42883E+00	0.23413E+00	0.10626E+00
-0.99376E-16	-0.42883E+00	-0.23413E+00	-0.10626E+00
0.68133E-16	-0.24133E+00	0.29527E+00	0.32338E+00
-0.61320E-16	-0.88674E-01	0.32863E+00	-0.36625E+00
-0.50000E+00	-0.30223E-16	0.25439E-16	0.19553E-16

MULTIPLE EIGENVALUE: -0.20000E+01

CORRESPONDING H(I) MATRIX FOR G1:

0.25000E+00	-.12500E+00	-.72022E-16	0.12500E+00	-.25000E+00	0.12500E+00	0.79446E-16	-.12500E+00
-.12500E+00	0.25000E+00	-.12500E+00	0.18869E-15	0.12500E+00	-.25000E+00	0.12500E+00	-.27876E-16
-.72022E-16	-.12500E+00	0.25000E+00	-.12500E+00	0.56607E-16	0.12500E+00	-.25000E+00	0.12500E+00
0.12500E+00	0.18869E-15	-.12500E+00	0.25000E+00	-.12500E+00	-.15989E-15	0.12500E+00	-.25000E+00
-.25000E+00	0.12500E+00	0.56607E-16	-.12500E+00	0.25000E+00	-.12500E+00	-.64031E-16	0.12500E+00
0.12500E+00	-.25000E+00	0.12500E+00	-.15989E-15	-.12500E+00	0.25000E+00	-.12500E+00	-.91895E-18
0.79446E-16	0.12500E+00	-.25000E+00	0.12500E+00	-.64031E-16	-.12500E+00	0.25000E+00	-.12500E+00
-.12500E+00	-.27876E-16	0.12500E+00	-.25000E+00	0.12500E+00	-.91895E-18	-.12500E+00	0.25000E+00
-.11868E-15	-.12500E+00	0.54601E-16	0.12500E+00	0.10740E-15	0.12500E+00	-.15208E-16	-.12500E+00
-.12500E+00	-.10061E-15	-.12500E+00	0.98557E-17	0.12500E+00	0.80950E-16	0.12500E+00	-.22152E-17
0.19750E-15	-.12500E+00	-.80778E-16	-.12500E+00	-.20270E-15	0.12500E+00	0.23049E-15	0.12500E+00
0.12500E+00	-.90233E-17	-.12500E+00	-.28161E-17	-.12500E+00	-.77367E-17	0.12500E+00	0.69488E-19
-.59012E-16	0.12500E+00	-.17887E-15	-.12500E+00	0.70294E-16	-.12500E+00	0.13947E-15	0.12500E+00
0.12500E+00	-.24326E-15	0.12500E+00	-.26107E-16	-.12500E+00	0.43984E-16	-.12500E+00	0.13466E-16
-.20463E-15	0.12500E+00	0.12327E-15	0.12500E+00	0.20984E-15	-.12500E+00	-.27299E-15	-.12500E+00
-.12500E+00	0.12133E-16	0.12500E+00	0.28663E-17	0.12500E+00	0.46274E-17	-.12500E+00	-.11962E-18

-.11863E-15	-.12500E+00	0.19750E-15	0.12500E+00	-.59012E-16	0.12500E+00	-.20463E-15	-.12500E+00
-.12500E+00	-.10061E-15	-.12500E+00	-.90233E-17	0.12500E+00	-.24326E-16	0.12500E+00	0.12133E-16
0.54601E-15	-.12500E+00	-.80778E-16	-.12500E+00	-.17887E-15	0.12500E+00	0.12327E-15	0.12500E+00
0.12500E+00	0.98557E-17	-.12500E+00	-.28161E-17	-.12500E+00	-.26107E-16	0.12500E+00	0.28663E-17
0.10740E-15	0.12500E+00	-.20270E-15	-.12500E+00	0.70294E-16	-.12500E+00	0.20984E-15	0.12500E+00
0.12500E+00	0.80950E-16	0.12500E+00	-.77367E-17	-.12500E+00	0.43984E-15	-.12500E+00	0.46274E-17
-.15208E-16	0.12500E+00	0.23049E-15	0.12500E+00	0.13947E-15	-.12500E+00	-.27299E-15	-.12500E+00
-.12500E+00	-.22152E-17	0.12500E+00	0.69488E-19	0.12500E+00	0.18466E-15	-.12500E+00	-.11962E-18
0.25000E+00	0.12500E+00	-.18735E-15	-.12500E+00	-.25000E+00	-.12500E+00	-.13678E-17	0.12500E+00
0.12500E+00	0.25000E+00	0.12500E+00	-.30911E-16	-.12500E+00	-.25000E+00	-.12500E+00	-.72436E-17
-.18735E-15	0.12500E+00	0.25000E+00	0.12500E+00	0.30452E-15	-.12500E+00	-.25000E+00	-.12500E+00
-.12500E+00	-.30911E-16	0.12500E+00	0.25000E+00	0.12500E+00	0.48001E-17	-.12500E+00	-.25000E+00
-.25000E+00	-.12500E+00	0.30452E-15	0.12500E+00	0.25000E+00	0.12500E+00	-.11580E-15	-.12500E+00
-.12500E+00	-.25000E+00	-.12500E+00	0.48001E-17	0.12500E+00	0.25000E+00	0.12500E+00	0.33355E-16
-.13678E-17	-.12500E+00	-.25000E+00	-.12500E+00	-.11580E-15	0.12500E+00	0.25000E+00	0.12500E+00
0.12500E+00	-.72436E-17	-.12500E+00	-.25000E+00	-.12500E+00	0.33355E-16	0.12500E+00	0.25000E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.25000E+00	0.12500E+00	0.12582E-15	-.64158E-16	0.36933E-16	-.12500E+00	-.12500E+00	-.12500E+00	
0.12500E+00	0.25000E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.28586E-16	-.25875E-16	-.55183E-16	
0.12582E-15	-.12500E+00	0.25000E+00	-.17551E-16	-.37467E-16	0.12500E+00	-.12500E+00	-.12500E+00	
-.64158E-16	-.12500E+00	-.17551E-16	0.25000E+00	0.32618E-16	-.12500E+00	0.12500E+00	-.12500E+00	
0.36933E-16	-.12500E+00	-.37467E-16	0.32618E-16	0.25000E+00	-.12500E+00	-.12500E+00	0.12500E+00	
-.12500E+00	-.28585E-16	0.12500E+00	-.12500E+00	-.12500E+00	0.25000E+00	0.18291E-16	-.52917E-16	
-.12500E+00	-.25875E-16	-.12500E+00	0.12500E+00	-.12500E+00	0.18291E-16	0.25000E+00	0.47673E-16	
-.12500E+00	-.55183E-16	-.12500E+00	-.12500E+00	0.12500E+00	-.52917E-16	0.47673E-16	0.25000E+00	
-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	
-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	
-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	
-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	
0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	
0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	
0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	
0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	

-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	
-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	
-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	
0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	
0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	
-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	
0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	
0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	
0.25000E+00	-.24814E-16	-.15083E-16	-.18979E-15	0.14506E-15	0.15670E-16	0.19492E-16	-.25000E+00	
-.24814E-16	0.25000E+00	0.23244E-15	-.12997E-15	0.10521E-15	0.57697E-16	-.25000E+00	-.29218E-16	
-.15083E-15	0.23244E-15	0.25000E+00	0.44366E-16	-.11854E-15	-.25000E+00	-.25282E-15	0.12351E-16	
-.18979E-15	-.12997E-15	0.44366E-16	0.25000E+00	-.25000E+00	0.11877E-15	0.13323E-16	-.98826E-17	
0.14505E-15	0.10521E-15	-.11854E-15	-.25000E+00	0.25000E+00	-.44591E-16	0.11446E-16	0.54615E-16	
0.15670E-16	0.57697E-16	-.25000E+00	0.11877E-15	-.44591E-16	0.25000E+00	-.37313E-16	-.12938E-16	
0.19492E-15	-.25000E+00	-.25282E-15	0.13323E-16	0.11446E-16	-.37313E-16	0.25000E+00	0.34539E-16	
-.25000E+00	-.29218E-16	0.12351E-16	-.98826E-17	0.54615E-16	-.12938E-16	0.34539E-16	0.25000E+00	

EIGENVALUE: -0.13551E-15

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.13861E+00	0.41023E+00	0.12992E-02	-0.12992E-02	0.41023E+00	0.13861E+00
0.38194E+00	0.19552E+00	0.58298E-01	0.58298E-01	-0.19552E+00	0.38194E+00
0.34152E+00	-0.37797E-01	0.26351E+00	-0.26351E+00	-0.37797E-01	-0.34152E+00
0.61271E-01	-0.38050E+00	-0.19740E+00	-0.19740E+00	0.38050E+00	0.61271E-01
-0.13861E+00	0.41023E+00	0.12992E-02	-0.12992E-02	0.41023E+00	0.13861E+00
0.38194E+00	0.19552E+00	0.58298E-01	0.58298E-01	-0.19552E+00	0.38194E+00
0.34152E+00	-0.37797E-01	0.26351E+00	-0.26351E+00	-0.37797E-01	-0.34152E+00
0.61271E-01	-0.38050E+00	-0.19740E+00	-0.19740E+00	0.38050E+00	0.61271E-01
-0.31775E+00	-0.26015E+00	0.13731E+00	-0.13731E+00	-0.26015E+00	0.31775E+00
-0.18878E+00	-0.65443E-01	0.38416E+00	0.38416E+00	0.65443E-01	-0.18878E+00
0.11484E+00	-0.11228E+00	-0.40213E+00	0.40213E+00	-0.11228E+00	-0.11484E+00
-0.25443E+00	0.25042E+00	-0.24506E+00	-0.24506E+00	-0.25042E+00	-0.25443E+00
-0.31775E+00	-0.26015E+00	0.13731E+00	-0.13731E+00	-0.26015E+00	0.31775E+00
-0.18878E+00	-0.65443E-01	0.38416E+00	0.38416E+00	0.65443E-01	-0.18878E+00
0.11484E+00	-0.11228E+00	-0.40213E+00	0.40213E+00	-0.11228E+00	-0.11484E+00
-0.25443E+00	0.25042E+00	-0.24506E+00	-0.24506E+00	-0.25042E+00	-0.25443E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.13383E+00	0.39738E+00	-0.77695E-01	-0.16823E+00	0.35502E+00	0.19697E+00
-0.13383E+00	-0.39738E+00	0.77695E-01	0.16823E+00	-0.35502E+00	-0.19697E+00
0.21384E+00	0.13197E+00	0.31970E+00	0.20353E+00	-0.41009E+00	-0.70748E-02
0.17010E-01	-0.18728E+00	-0.52921E+00	0.77340E-01	-0.11733E+00	-0.19956E+00
-0.36469E+00	-0.34207E+00	0.28721E+00	-0.11264E+00	0.17239E+00	0.96616E-02
-0.21384E+00	-0.13197E+00	-0.31970E+00	-0.20353E+00	0.41009E+00	0.70748E-02
-0.17010E-01	0.18728E+00	0.52921E+00	-0.77340E-01	0.11733E+00	0.19956E+00
0.36469E+00	0.34207E+00	-0.28721E+00	0.11264E+00	-0.17239E+00	-0.96616E-02
-0.21976E-02	0.20160E-01	-0.22539E-01	-0.54292E+00	-0.28161E+00	-0.41835E-02
-0.44667E+00	0.27784E+00	-0.85377E-01	0.28229E+00	-0.71320E-01	0.78921E-01
0.18689E+00	0.10021E-01	0.14222E+00	0.90347E-01	0.26487E+00	-0.49135E+00
0.26197E+00	-0.30802E+00	-0.34306E-01	0.17028E+00	0.88060E-01	0.41661E+00
0.26197E+00	-0.30802E+00	-0.34306E-01	0.17028E+00	0.88060E-01	0.41661E+00
0.18689E+00	0.10021E-01	0.14222E+00	0.90347E-01	0.26487E+00	-0.49135E+00
-0.44667E+00	0.27784E+00	-0.85377E-01	0.28229E+00	-0.71320E-01	0.78921E-01
-0.21976E-02	0.20160E-01	-0.22539E-01	-0.54292E+00	-0.28161E+00	-0.41835E-02

MULTIPLE EIGENVALUE: -0.13551E-15

CORRESPONDING H(I) MATRIX FOR G1:

0.37500E+00	-.19323E-15	-.12500E+00	0.13693E-15	0.37500E+00	-.19884E-15	-.12500E+00	0.13362E-15
-.19328E-15	0.37500E+00	0.23017E-15	-.12500E+00	-.19578E-15	0.37500E+00	0.21443E-15	-.12500E+00
-.12500E+00	0.23017E-15	0.37500E+00	0.41458E-16	-.12500E+00	0.20033E-15	0.37500E+00	0.60723E-16
0.13693E-15	-.12500E+00	0.41458E-16	0.37500E+00	0.13758E-15	-.12500E+00	0.18593E-16	0.37500E+00
0.37500E+00	-.19573E-15	-.12500E+00	0.13758E-15	0.37500E+00	-.20134E-15	-.12500E+00	0.13427E-15
-.19684E-15	0.37500E+00	0.20033E-15	-.12500E+00	-.20134E-15	0.37500E+00	0.18459E-15	-.12500E+00
-.12500E+00	0.21443E-15	0.37500E+00	0.18593E-16	-.12500E+00	0.18459E-15	0.37500E+00	0.37858E-16
0.13362E-15	-.12500E+00	0.60723E-16	0.37500E+00	0.13427E-15	-.12500E+00	0.37858E-16	0.37500E+00
-.12500E+00	-.92125E-16	-.12500E+00	-.12303E-15	-.12500E+00	-.56384E-16	-.12500E+00	-.12503E-15
0.46760E-16	-.12500E+00	-.15438E-15	-.12500E+00	0.67861E-16	-.12500E+00	-.15609E-15	-.12500E+00
-.12500E+00	0.56993E-15	-.12500E+00	-.35889E-16	-.12500E+00	0.56639E-16	-.12500E+00	-.49842E-16
-.54957E-16	-.12500E+00	-.68755E-16	-.12500E+00	-.74210E-16	-.12500E+00	-.28443E-16	-.12500E+00
-.12500E+00	-.11252E-15	-.12500E+00	-.12917E-15	-.12500E+00	-.76777E-16	-.12500E+00	-.13117E-15
0.54454E-16	-.12500E+00	-.17334E-15	-.12500E+00	0.75555E-16	-.12500E+00	-.17505E-15	-.12500E+00
-.12500E+00	0.65006E-16	-.12500E+00	-.40319E-16	-.12500E+00	0.64652E-16	-.12500E+00	-.54272E-16
-.45339E-16	-.12500E+00	-.92453E-16	-.12500E+00	-.64592E-16	-.12500E+00	-.52140E-16	-.12500E+00

-.12500E+00	0.46760E-16	-.12500E+00	-.54957E-16	-.12500E+00	0.54454E-15	-.12500E+00	-.45339E-16
-.92125E-16	-.12500E+00	0.56993E-16	-.12500E+00	-.11252E-15	-.12500E+00	0.65006E-16	-.12500E+00
-.12500E+00	-.15438E-15	-.12500E+00	-.68755E-16	-.12500E+00	-.17334E-15	-.12500E+00	-.92453E-16
-.12303E-15	-.12500E+00	-.35889E-16	-.12500E+00	-.12917E-15	-.12500E+00	-.40319E-16	-.12500E+00
-.12500E+00	0.67861E-16	-.12500E+00	-.74210E-16	-.12500E+00	0.75555E-16	-.12500E+00	-.64592E-16
-.56384E-16	-.12500E+00	0.56639E-16	-.12500E+00	-.76777E-16	-.12500E+00	0.64652E-16	-.12500E+00
-.12500E+00	-.15609E-15	-.12500E+00	-.28443E-16	-.12500E+00	-.17505E-15	-.12500E+00	-.52140E-16
-.12503E-15	-.12500E+00	-.49842E-16	-.12500E+00	-.13117E-15	-.12500E+00	-.54272E-16	-.12500E+00
0.37500E+00	-.10634E-16	-.12500E+00	0.20982E-15	0.37500E+00	0.70044E-17	-.12500E+00	0.23187E-15
-.10634E-16	0.37500E+00	0.11340E-15	-.12500E+00	0.51763E-17	0.37500E+00	0.10987E-15	-.12500E+00
-.12500E+00	0.11340E-15	0.37500E+00	-.10246E-15	-.12500E+00	0.10702E-15	0.37500E+00	-.11043E-15
0.20982E-15	-.12500E+00	-.10246E-15	0.37500E+00	0.22054E-15	-.12500E+00	-.10252E-15	0.37500E+00
0.37500E+00	0.51763E-17	-.12500E+00	0.22054E-15	0.37500E+00	0.22815E-16	-.12500E+00	0.24259E-15
0.70044E-17	0.37500E+00	0.10702E-15	-.12500E+00	0.22815E-16	0.37500E+00	0.10350E-15	-.12500E+00
-.12500E+00	0.10987E-15	0.37500E+00	-.10252E-15	-.12500E+00	0.10350E-15	0.37500E+00	-.11049E-15
0.23187E-15	-.12500E+00	-.11043E-15	0.37500E+00	0.24259E-15	-.12500E+00	-.11049E-15	0.37500E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.37500E+00	-.37500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00
-.37500E+00	0.37500E+00	0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00
-.12500E+00	0.12500E+00	0.37500E+00	-.12500E+00	-.12500E+00	-.37500E+00	0.12500E+00	0.12500E+00
-.12500E+00	0.12500E+00	-.12500E+00	0.37500E+00	-.12500E+00	0.12500E+00	-.37500E+00	0.12500E+00
-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00	0.12500E+00	0.12500E+00	-.37500E+00
0.12500E+00	-.12500E+00	-.37500E+00	0.12500E+00	0.12500E+00	0.37500E+00	-.12500E+00	-.12500E+00
0.12500E+00	-.12500E+00	0.12500E+00	-.37500E+00	0.12500E+00	-.12500E+00	0.37500E+00	-.12500E+00
0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.37500E+00	-.12500E+00	-.12500E+00	0.37500E+00
0.86408E-16	-.11341E-15	-.63718E-16	-.19977E-16	0.72509E-17	0.30837E-16	-.23350E-16	0.43281E-16
0.14649E-15	-.21579E-15	0.96786E-16	0.23757E-15	-.52224E-15	-.43848E-16	-.20967E-15	0.44299E-15
-.33361E-17	-.37858E-17	-.90835E-16	-.33359E-15	0.31178E-15	-.10868E-15	0.31749E-15	-.29004E-15
-.22019E-15	0.32361E-15	0.48149E-16	0.10347E-15	0.21598E-15	0.13130E-15	-.71936E-16	-.20901E-15
-.27374E-15	0.37716E-15	-.30321E-16	0.15928E-15	0.29219E-15	0.20977E-15	-.12775E-15	-.28521E-15
-.84440E-17	0.13221E-17	-.11524E-15	-.35339E-15	0.36110E-15	-.84266E-16	0.33729E-15	-.33936E-15
0.25763E-15	-.32693E-15	0.99010E-16	0.11913E-15	-.51717E-15	-.46072E-16	-.91233E-16	0.43792E-15
0.91149E-16	-.11815E-15	-.39007E-16	-.20829E-16	-.21349E-16	0.61253E-17	-.22498E-16	0.71880E-16

0.86403E-15	0.14649E-15	-.33361E-17	-.22019E-15	-.27374E-15	-.84440E-17	0.25763E-15	0.91149E-16
-.11341E-15	-.21579E-15	-.37858E-17	0.32361E-15	0.37716E-15	0.13221E-17	-.32693E-15	-.11815E-15
-.63713E-15	0.96786E-16	-.90835E-16	0.48149E-16	-.30321E-16	-.11524E-15	0.99010E-16	-.39007E-16
-.19977E-16	0.23757E-15	-.33359E-15	0.10347E-15	0.15928E-15	-.35339E-15	0.11913E-15	-.20829E-16
0.72503E-17	-.52224E-15	0.31178E-15	0.21598E-15	0.29219E-15	0.36110E-15	-.51717E-15	-.21349E-16
0.30837E-15	-.43848E-16	-.10868E-15	0.13130E-15	0.20977E-15	-.84266E-16	-.46072E-16	0.61253E-17
-.23350E-16	-.20967E-15	0.31749E-15	-.71936E-16	-.12775E-15	0.33729E-15	-.91233E-16	-.22498E-16
0.43281E-15	0.44299E-15	-.29004E-15	-.20901E-15	-.28521E-15	-.33936E-15	0.43792E-15	0.71880E-16
0.37500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00
-.12500E+00	0.37500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00	-.12500E+00
-.12500E+00	-.12500E+00	0.37500E+00	-.12500E+00	-.12500E+00	0.37500E+00	-.12500E+00	-.12500E+00
-.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00	0.37500E+00	-.12500E+00	-.12500E+00	-.12500E+00
-.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00	0.37500E+00	-.12500E+00	-.12500E+00	-.12500E+00
-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00	-.12500E+00	-.12500E+00
-.12500E+00	0.37500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00	-.12500E+00
0.37500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.37500E+00

EIGENVALUE: 0.20000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

0.11128E+00	-0.14740E+00	0.39165E+00	-0.25000E+00
0.35189E-01	0.27829E+00	0.41390E+00	-0.13955E-16
0.35760E+00	0.20370E+00	0.13464E+00	0.25000E+00
0.24632E+00	0.35110E+00	-0.25701E+00	0.58141E-16
-0.11128E+00	0.14740E+00	-0.39165E+00	0.25000E+00
-0.35189E-01	-0.27829E+00	-0.41390E+00	-0.17641E-16
-0.35760E+00	-0.20370E+00	-0.13464E+00	-0.25000E+00
-0.24632E+00	-0.35110E+00	0.25701E+00	-0.61603E-16
-0.32241E+00	0.74593E-01	0.27926E+00	0.25000E+00
0.43369E+00	-0.22199E+00	0.11239E+00	0.13544E-15
-0.76090E-01	0.42569E+00	0.22251E-01	-0.25000E+00
-0.20209E-15	0.41706E-16	-0.20643E-16	0.50000E+00
0.32241E+00	-0.74593E-01	-0.27926E+00	-0.25000E+00
-0.43369E+00	0.22199E+00	-0.11239E+00	-0.18579E-15
0.76090E-01	-0.42569E+00	-0.22251E-01	0.25000E+00
0.11676E-15	-0.42932E-17	-0.24533E-17	-0.50000E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

0.21135E+00	0.18268E+00	-0.33085E+00	0.25000E+00
0.19151E+00	-0.30796E+00	-0.23661E+00	0.25000E+00
-0.19151E+00	0.30796E+00	0.23661E+00	0.25000E+00
0.30014E+00	0.27203E+00	0.15302E+00	-0.25000E+00
-0.28031E+00	0.21861E+00	-0.24726E+00	-0.25000E+00
-0.21135E+00	-0.18268E+00	0.33085E+00	0.25000E+00
0.28031E+00	-0.21861E+00	0.24726E+00	-0.25000E+00
-0.30014E+00	-0.27203E+00	-0.15302E+00	-0.25000E+00
0.33598E-16	0.37761E-15	0.14612E-15	0.50000E+00
0.49165E+00	-0.35931E-01	-0.83585E-01	0.88647E-17
-0.88793E-01	-0.89353E-01	-0.48387E+00	0.78259E-16
0.19835E-01	0.49064E+00	-0.94242E-01	-0.37912E-15
-0.19835E-01	-0.49064E+00	0.94242E-01	0.36834E-15
0.88793E-01	0.89353E-01	0.48387E+00	-0.14385E-15
-0.49165E+00	0.35931E-01	0.83585E-01	-0.34390E-16
0.45277E-17	-0.45360E-15	0.21288E-16	-0.50000E+00

MULTIPLE EIGENVALUE: 0.20000E+01

CORRESPONDING H(I) MATRIX FOR G1:

0.25000E+00	0.12500E+00	-.52454E-16	-.12500E+00	-.25000E+00	-.12500E+00	0.65313E-16	0.12500E+00
0.12500E+00	0.25000E+00	0.12500E+00	0.11850E-15	-.12500E+00	-.25000E+00	-.12500E+00	0.42311E-16
-.52454E-16	0.12500E+00	0.25000E+00	0.12500E+00	0.45845E-16	-.12500E+00	-.25000E+00	-.12500E+00
-.12500E+00	0.11850E-15	0.12500E+00	0.25000E+00	0.12500E+00	-.36136E-16	-.12500E+00	-.25000E+00
-.25000E+00	-.12500E+00	0.45845E-16	0.12500E+00	0.25000E+00	0.12500E+00	-.58705E-16	-.12500E+00
-.12500E+00	-.25000E+00	-.12500E+00	-.86136E-16	0.12500E+00	0.25000E+00	0.12500E+00	-.74673E-16
0.65313E-16	-.12500E+00	-.25000E+00	-.12500E+00	-.58705E-16	0.12500E+00	0.25000E+00	0.12500E+00
0.12500E+00	0.42311E-16	-.12500E+00	-.25000E+00	-.12500E+00	-.74673E-16	0.12500E+00	0.25000E+00
-.80182E-16	0.12500E+00	-.19131E-16	-.12500E+00	0.79715E-16	-.12500E+00	0.62399E-16	0.12500E+00
0.12500E+00	-.13080E-15	0.12500E+00	-.46435E-16	-.12500E+00	0.10959E-15	-.12500E+00	0.54076E-16
0.16980E-15	0.12500E+00	0.19396E-16	0.12500E+00	-.18164E-15	-.12500E+00	0.13063E-15	-.12500E+00
-.12500E+00	-.11027E-15	0.12500E+00	-.76049E-18	0.12500E+00	-.47713E-17	-.12500E+00	-.97048E-18
-.10642E-15	-.12500E+00	-.10353E-15	0.12500E+00	0.10689E-15	0.12500E+00	0.60264E-16	-.12500E+00
-.12500E+00	0.17358E-15	-.12500E+00	0.23051E-16	0.12500E+00	0.38595E-17	0.12500E+00	-.30691E-16
-.18041E-15	-.12500E+00	0.26570E-16	-.12500E+00	0.19224E-15	0.12500E+00	-.17660E-15	0.12500E+00
0.12500E+00	0.88759E-17	-.12500E+00	-.11864E-17	-.12500E+00	0.69219E-17	0.12500E+00	0.29174E-17

-.80182E-15	0.12500E+00	0.16980E-15	-.12500E+00	-.10642E-15	-.12500E+00	-.18041E-15	0.12500E+00
0.12500E+00	-.13080E-15	0.12500E+00	-.11027E-16	-.12500E+00	0.17358E-16	-.12500E+00	0.88759E-17
-.19131E-15	0.12500E+00	0.19396E-16	0.12500E+00	-.10353E-15	-.12500E+00	0.26570E-16	-.12500E+00
-.12500E+00	-.46435E-16	0.12500E+00	-.76049E-18	0.12500E+00	0.23051E-16	-.12500E+00	-.11864E-17
0.79715E-15	-.12500E+00	-.18164E-15	0.12500E+00	0.10689E-15	0.12500E+00	0.19224E-15	-.12500E+00
-.12500E+00	0.10959E-15	-.12500E+00	-.47713E-17	0.12500E+00	0.38595E-17	0.12500E+00	0.69219E-17
0.62399E-15	-.12500E+00	0.13063E-15	-.12500E+00	0.60264E-16	0.12500E+00	-.17660E-15	0.12500E+00
0.12500E+00	0.54076E-16	-.12500E+00	-.97048E-18	-.12500E+00	-.30691E-16	0.12500E+00	0.29174E-17
0.25000E+00	-.12500E+00	-.20775E-15	0.12500E+00	-.25000E+00	0.12500E+00	0.22496E-16	-.12500E+00
-.12500E+00	0.25000E+00	-.12500E+00	-.31503E-16	0.12500E+00	-.25000E+00	0.12500E+00	-.16405E-16
-.20775E-15	-.12500E+00	0.25000E+00	-.12500E+00	0.32113E-15	0.12500E+00	-.25000E+00	0.12500E+00
0.12500E+00	-.31503E-16	-.12500E+00	0.25000E+00	-.12500E+00	0.63268E-17	0.12500E+00	-.25000E+00
-.25000E+00	0.12500E+00	0.32113E-15	-.12500E+00	0.25000E+00	-.12500E+00	-.13588E-15	0.12500E+00
0.12500E+00	-.25000E+00	0.12500E+00	0.63268E-17	-.12500E+00	0.25000E+00	-.12500E+00	0.41581E-16
0.22496E-16	0.12500E+00	-.25000E+00	0.12500E+00	-.13588E-15	-.12500E+00	0.25000E+00	-.12500E+00
-.12500E+00	-.16405E-16	0.12500E+00	-.25000E+00	0.12500E+00	0.41581E-16	-.12500E+00	0.25000E+00

CORRESPONDING H(I) MATRIX FOR G2:

0.25000E+00	0.12500E+00	-.71976E-16	0.67962E-16	-.31163E-16	-.12500E+00	-.12500E+00	-.12500E+00
0.12500E+00	0.25000E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.10466E-17	-.76802E-16	-.67912E-16
-.71976E-16	-.12500E+00	0.25000E+00	-.91078E-16	-.61637E-16	0.12500E+00	-.12500E+00	-.12500E+00
0.67962E-16	-.12500E+00	-.91078E-16	0.25000E+00	-.17204E-16	-.12500E+00	0.12500E+00	-.12500E+00
-.31163E-16	-.12500E+00	-.61637E-16	-.17204E-16	0.25000E+00	-.12500E+00	-.12500E+00	0.12500E+00
-.12500E+00	-.10466E-17	0.12500E+00	-.12500E+00	-.12500E+00	0.25000E+00	-.37374E-16	0.10737E-15
-.12500E+00	-.76802E-16	-.12500E+00	0.12500E+00	-.12500E+00	-.37374E-16	0.25000E+00	0.68040E-16
-.12500E+00	-.67912E-15	-.12500E+00	-.12500E+00	0.12500E+00	0.10737E-15	0.68040E-16	0.25000E+00
0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00
0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00
0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00
0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00
-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00
-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00
-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00
-.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00

0.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00
0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	-.12500E+00
0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00
-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00
-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00
0.12500E+00	-.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00
-.12500E+00	0.12500E+00	-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00
-.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00	0.12500E+00	-.12500E+00	0.12500E+00	0.12500E+00
0.25000E+00	-.48305E-17	-.68299E-16	-.17395E-16	0.12007E-16	0.35505E-16	-.79320E-17	-.25000E+00
-.48305E-17	0.25000E+00	0.26439E-15	0.90902E-17	-.15050E-15	-.21555E-16	-.25000E+00	0.12313E-16
-.68299E-15	0.26439E-15	0.25000E+00	0.49456E-16	-.85964E-16	-.25000E+00	0.86373E-16	-.93013E-17
-.17395E-16	0.90902E-17	0.49456E-16	0.25000E+00	-.25000E+00	-.27999E-16	0.12317E-15	-.34913E-16
0.12007E-15	-.15050E-15	-.85964E-16	-.25000E+00	0.25000E+00	0.64507E-16	0.18242E-16	0.40301E-16
0.35505E-16	-.21555E-16	-.25000E+00	-.27999E-16	0.64507E-16	0.25000E+00	-.32921E-15	0.42095E-16
-.79320E-17	-.25000E+00	0.86373E-16	0.12317E-15	0.18242E-16	-.32921E-15	0.25000E+00	0.44956E-18
-.25000E+00	0.12313E-16	-.93013E-17	-.34913E-16	0.40301E-16	0.42095E-16	0.44956E-18	0.25000E+00

EIGENVALUE: 0.40000E+01

CORRESPONDING BASIS EIGENVECTORS FOR G1 ARE:

-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00

CORRESPONDING BASIS EIGENVECTORS FOR G2 ARE:

-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00
-0.25000E+00

APPENDIX II

Automorphism Group Generators For The Eight 25-Point Strongly Regular Graphs

Graph 1:

1	2	3	4	5	6	7	11	13	16	19	21	25
2	3	4	5	7	8	12	14	17	20	21	22	
3	4	5	8	9	13	15	16	18	22	23		
4	5	9	10	11	14	17	19	23	24			
5	6	10	12	15	18	20	24	25				
6	7	8	9	10	11	12	16	18	21	24		
7	8	9	10	12	13	17	19	22	25			
8	9	10	13	14	18	20	21	23				
9	10	14	15	16	19	22	24					
10	11	15	17	20	23	25						
11	12	13	14	15	16	17	21	23				
12	13	14	15	17	18	22	24					
13	14	15	18	19	23	25						
14	15	19	20	21	24							
15	16	20	22	25								
16	17	18	19	20	21	22						
17	18	19	20	22	23							
18	19	20	23	24								
19	20	24	25									
20	21	25										
21	22	23	24	25								
22	23	24	25									
23	24	25										
24	25											
25												

Automorphism Group Order = 624

Generators:

γ_1 = (1)(2)(3)(4)(5)(6 25)(7 21)(8 22)(9 23)(10 24)(11 19)(12 20)
(13 16)(14 17)(15 18)

γ_2 = (1)(2 3 5 4)(6 11 21 16)(7 13 25 19)(8 15 24 17)(9 12 23 20)
(10 14 22 18)

γ_3 = (1)(2 6 25)(3 11 19)(4 16 13)(5 21 7)(8 10 20)(12 24 22)(14 9 15)
(17 18 23)

γ_4 = (1 2 3 4 5)(6 7 8 9 10)(11 12 13 14 15)(16 17 18 19 20)
(21 22 23 24 25)

Graph 2:

1	2	3	4	5	6	7	8	9	10	11	12	13
2	3	4	7	8	11	14	15	16	17	21	25	
3	5	6	9	11	14	15	16	18	20	24		
4	5	7	8	13	15	17	18	19	20	23		
5	6	9	13	16	17	18	19	22	25			
6	7	9	10	14	17	20	21	22	23			
7	8	10	16	19	20	21	22	24				
8	9	12	14	18	22	23	24	25				
9	12	15	19	21	23	24	25					
10	11	12	13	14	16	17	19	23	24			
11	12	13	17	18	20	21	24	25				
12	13	14	15	18	19	21	22					
13	15	16	20	22	23	25						
14	15	16	17	18	22	23						
15	16	19	20	21	23							
16	19	22	24	25								
17	18	19	21	23	25							
18	19	20	22	24								
19	21	24										
20	21	22	23	24								
21	22	25										
22	25											
23	24	25										
24	25											
25												

Automorphism Group Order = 72

Generators:

γ_1 = (1)(2)(3)(4 7 8)(5 6 9)(10 12 13)(11)(14 15 16)(17 21 25)
(18 20 24)(19 22 23)

γ_2 = (1)(2)(3 11)(4 7)(5 10)(6 13)(8)(9 12)(14 25)(15 21)(16 17)
(18 24)(19)(20)(22 23)

γ_3 = (1)(2 3 11)(4 5 13)(6 10 7)(8 9 12)(14 24 21)(15 18 25)
(16 20 17)(19 22 23)

γ_4 = (1)(2 4 7 8)(3 13 6 12)(5 10 9 11)(14 15 20 22)(16 23 21 18)
(19 24 25 17)

Graph 3:

1	2	3	4	5	6	7	8	9	10	11	12	13
2	3	4	11	12	13	14	15	16	17	18	19	
3	4	5	6	7	14	16	19	20	22	24		
4	5	6	7	15	17	18	21	23	25			
5	9	10	13	14	17	20	22	23	25			
6	8	10	12	15	16	21	22	23	24			
7	8	9	11	18	19	20	21	24	25			
8	9	10	13	14	16	17	18	21	24			
9	10	12	15	16	17	19	20	25				
10	11	14	15	18	19	22	23					
11	12	13	18	19	20	21	22	23				
12	13	15	16	20	23	24	25					
13	14	17	21	22	24	25						
14	16	18	19	23	24	25						
15	17	18	19	22	24	25						
16	17	19	20	21	23							
17	18	20	21	22								
18	20	23	24									
19	21	22	25									
20	22	23	24									
21	22	23	25									
22	24											
23	25											
24	25											
25												

Automorphism Group Order = 6

Generators:

γ_1 = (1)(2)(3)(4)(5 6 7)(8 9 10)(11 13 12)(14 16 19)(15 18 17)
(20 22 24)(21 25 23)

γ_2 = (1)(2)(3 4)(5 6)(7)(8 9)(10)(11)(12 13)(14 15)(16 17)(18 19)
(20 21)(22 23)(24 25)

Graph 4:

1	2	3	4	5	6	7	8	9	10	20	21	22
2	6	7	8	10	12	13	14	16	17	18	20	
3	5	7	8	9	11	13	15	16	18	19	21	
4	5	6	9	10	11	12	14	15	17	19	22	
5	6	7	12	13	15	18	22	23	24			
6	7	11	13	14	19	20	24	25				
7	11	12	16	17	21	23	25					
8	9	10	12	13	14	19	21	23	24			
9	10	11	13	16	17	22	24	25				
10	11	12	15	18	20	23	25					
11	14	16	18	19	23	25						
12	15	16	17	19	23	24						
13	14	15	17	18	24	25						
14	17	18	19	21	22	23						
15	17	18	19	20	21	25						
16	17	18	19	20	22	24						
17	21	22	25									
18	20	22	23									
19	20	21	24									
20	21	22	24									
21	22	23	25									
22	23	24										
23	24	25										
24	25											
25												

Automorphism Group Order = 3

Generator:

$$\gamma_1 = (1)(2\ 3\ 4)(5\ 6\ 7)(8\ 9\ 10)(11\ 12\ 13)(14\ 16\ 15)(17\ 18\ 19)(20\ 21\ 22)(23\ 24\ 25)$$

Graph 5:

1	8	9	10	11	12	13	20	21	22	23	24	25
2	3	4	8	9	11	13	15	17	18	19	20	22
3	4	9	10	12	13	14	15	16	17	21	23	
4	8	10	11	12	14	16	18	19	24	25		
5	6	7	8	10	11	12	15	16	17	18	20	23
6	7	8	9	11	13	14	16	17	19	21	25	
7	9	10	12	13	14	15	18	19	22	24		
8	11	12	13	14	15	20	25					
9	11	12	13	16	18	21	22					
10	11	12	13	17	19	23	24					
11	16	17	22	24								
12	15	18	21	25								
13	14	19	20	23								
14	15	16	22	23	24	25						
15	17	20	21	22	24							
16	18	20	21	23	24							
17	19	21	22	23	25							
18	19	20	22	23	25							
19	20	21	24	25								
20	21	23	24									
21	24	25										
22	23	24	25									
23	25											
24												
25												

Automorphism Group Order = 6

Generators:

γ_1 = (1)(2)(3 4)(5 7)(6)(8 9)(10)(11 13)(12)(14 16)
(15 18)(17 19)(20 22)(21 25)(23 24)

γ_2 = (1)(2 3 4)(5 6 7)(8 9 10)(11 13 12)(14 18 17)(15 16 19)
(20 21 24)(22 23 25)

Graph 6:

1	2 5 8 9 10 11 14 15 20 21 22 23
2	3 4 5 8 9 12 13 20 21 24 25
3	5 6 7 12 13 16 17 20 21 22 23
4	5 6 7 8 9 14 15 16 17 24 25
5	6 7 10 11 22 23 24 25
6	7 8 10 11 13 15 17 19 20
7	9 10 11 12 14 16 18 21
8	9 10 12 13 15 16 19 22
9	11 12 13 14 17 18 23
10	13 14 18 19 21 22 25
11	12 15 18 19 20 23 24
12	16 18 19 20 22 25
13	17 18 19 21 23 24
14	16 17 19 20 21 23 25
15	16 17 18 20 21 22 24
16	19 21 22 23 24
17	18 20 22 23 25
18	21 22 24 25
19	20 23 24 25
20	21 25
21	24
22	23 25
23	24
24	25
25	

Automorphism Group Order = 2

Generator:

γ_1 = (1)(2)(3)(4)(5)(6 7)(8 9)(10 11)(12 13)(14 15)(16 17)(18 19)
(20 21)(22 23)(24 25)

Graph 7:

1	2	3	4	5	8	9	12	13	16	17	22	23
2	5	6	7	12	13	14	15	20	21	22	23	
3	5	10	11	12	13	14	15	16	17	18	19	
4	5	8	9	10	11	14	15	22	23	24	25	
5	8	9	14	15	18	19	20	21				
6	7	8	9	10	11	13	15	16	19	21	23	
7	8	9	10	11	12	14	17	18	20	22		
8	9	10	13	17	18	21	24					
9	11	12	16	19	20	25						
10	12	14	15	16	18	23	24					
11	13	14	15	17	19	22	25					
12	13	14	16	20	24	25						
13	15	17	21	24	25							
14	19	21	22	24								
15	18	20	23	25								
16	18	19	21	22	23	25						
17	18	19	20	22	23	24						
18	20	21	22	25								
19	20	21	23	24								
20	23	24	25									
21	22	24	25									
22	23	25										
23	24											
24	25											
25												

Automorphism Group Order = 2

Generator:

γ_1 = (1)(2)(3)(4)(5)(6 7)(8 9)(10 11)(12 13)(14 15)(16 17)(18 19)
(20 21)(22 23)(24 25)

Graph 8:

1	4	9	10	11	12	13	16	17	19	23	24	25
2	6	9	10	11	14	15	16	18	20	22	23	24
3	4	5	7	11	14	15	16	19	20	21	24	25
4	7	11	12	14	16	17	18	21	22	23		
5	8	10	11	13	15	17	19	20	21	22	23	
6	7	8	12	13	14	16	19	20	22	23	25	
7	12	13	15	17	18	20	22	24	25			
8	9	12	14	15	17	18	19	21	23	25		
9	11	13	14	17	18	20	21	24	25			
10	12	13	15	16	18	19	21	22	24			
11	18	19	20	22	23	25						
12	18	19	20	21	23	24						
13	16	17	20	21	22	25						
14	16	17	19	21	22	24						
15	16	17	18	23	24	25						
16	21	23	25									
17	22	23	24									
18	21	22	25									
19	22	24	25									
20	21	23	24									
21												
22												
23												
24												
25												

Automorphism Group Order = 1

Generator:

γ_1 = (1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)(16)(17)
(18)(19)(20)(21)(22)(23)(24)(25)