USE OF THE SPECTRUM IN GRAPH ISOMORPHISM

by

Rachel Gelbart

B.A., Tel Aviv University, 1972

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in the Department
of
Computer Science

We accept this thesis as conforming to the
required standard

THE UNIVERSITY OF BRITISH COLUMBIA
February 1976
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date March 22, 1976
The following is a study of the use of the eigenvalues and eigenvectors of the adjacency matrices of graphs to determine graph isomorphism. Two randomly chosen graphs will in general have different eigenvalues, and comparing the eigenvalues provides a first filter when checking for isomorphism. If the graphs are cospectral, we propose an algorithm that compares their eigenvectors in order to find an isomorphism between them, if any exists. The algorithm involves backtracking, and it is therefore difficult to evaluate its performance. It was fast for all pairs of graphs we compared, except one. For graphs whose eigenvalues are all simple we present an efficient algorithm to generate the group of a graph.
### TABLE OF CONTENTS

**CHAPTER I** INTRODUCTION .................................................. 1

1.1 Definitions and Notation ................................................. 2

**CHAPTER II** GRAPH ISOMORPHISM ........................................... 5

2.1 Introduction ............................................................. 6

2.2 Heuristic methods ...................................................... 6

2.3 Cornelia's algorithm .................................................... 10

2.4 Algorithms for special graphs ......................................... 11

**CHAPTER III** THE SPECTRUM OF GRAPHS ..................................... 21

3.1 Introduction ............................................................. 21

3.2 Research on the spectrum of a graph ................................ 21

3.3 Algebraic properties ................................................... 27

3.4 Computation of eigenvalues and eigenvectors ....................... 30

**CHAPTER IV** THE EIGENVECTOR CONNECTION ............................. 32

4.1 Intuitive description of the algorithm ................................ 32

4.2 Connection with the group of the graph ............................. 43

4.3 Formal description of the Algorithm ................................ 50

4.4 Order of the computation .............................................. 58

4.5 Conclusion .............................................................. 64

**BIBLIOGRAPHY** ................................................................. 65

**APPENDIX A** Examples of cospectral graphs ............................ 69

**APPENDIX B** A nasty pair of cospectral graphs ......................... 77
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.1 Smallest cospectral graphs</td>
<td>17</td>
</tr>
<tr>
<td>2.5.2 Smallest cospectral connected graphs</td>
<td>17</td>
</tr>
<tr>
<td>4.1.1 Two cospectral isomorphic graphs</td>
<td>33</td>
</tr>
<tr>
<td>4.1.2 Smallest cospectral non-isomorphic trees</td>
<td>36</td>
</tr>
<tr>
<td>4.1.3 An 8-point graph with simple eigenvalues</td>
<td>37</td>
</tr>
<tr>
<td>4.1.4 Two cospectral isomorphic graphs</td>
<td>41</td>
</tr>
<tr>
<td>4.2.1 A 4-point graph</td>
<td>46</td>
</tr>
<tr>
<td>4.5.1 A 5-point graph with simple eigenvalues</td>
<td>59</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT

I would like to thank Dr. Abbe Mowshowitz for suggesting this topic and for his careful editing of the thesis.

I would also like to thank my friends, particularly G. I. McCalla, for their help, and my husband for his encouragement throughout my years in school.

Financial assistance was gratefully received from the Transportation Development Agency and the Department of Computer Science.
This thesis discusses a method for the detection of isomorphism between graphs that uses the eigenvalues and eigenvectors of the adjacency matrices of the graphs. The method is general purpose - it can be used for all types of undirected graphs. The basic idea is to map eigenvectors of the first graph onto eigenvectors of the second graph in order to find an isomorphism, if any exists. For graphs whose adjacency matrices have all distinct eigenvalues we present a method for finding the group of the graph. Although it is hard to evaluate the performance of this algorithm, because tentative solutions may be generated at an early stage and later rejected, the examples we have looked at suggest that the number of computation steps needed to find the group of the graph in this case is bounded above by a polynomial in the order of the group. For graphs whose eigenvalues are not simple, we present an algorithm to find one isomorphism (if any exists) between the two graphs. Here again it is difficult to give a general estimate of the number of computation steps required by the algorithm, since a large amount of backtracking may be necessary. This however was the case for only one of the pairs of graphs we tested (see Appendix B).

Most of the relevant definitions are presented in the next section, the rest are introduced as necessary in the following Chapters. In Chapter II, different algorithms for graph isomorphism are reviewed. Chapter III presents research on the spectrum of graphs which is relevant to our algorithm. In Chapter IV the algorithm is described, and its relation with the
group of the graph is discussed. Several examples illustrating the algorithms are given in Appendix A and Appendix B.

1.1 Definitions and Notation

The definitions and notation used here are largely those of Harary [15].

A graph $G = (V, E)$ consists of a finite non-empty set $V = V(G)$ together with a set $E = E(G)$ of unordered pairs of distinct elements of $V$. The elements of $V$ are called nodes (or points or vertices) of $G$, the elements of $E$ are called lines (or edges) of $G$.

If $x, y \in V$, $(x, y) \in E$, then $x$ and $y$ are adjacent. If $p, q \in V$, $x = (p, q) \in E$, $p$ and $q$ are incident to $x$.

A graph as defined here is finite, undirected, and has no loops or multiple edges. This restriction to undirected graphs is significant, since our algorithm uses a result (Theorem 2.1) which is valid only for symmetric matrices, and therefore holds only for the adjacency matrix of an undirected graph.

The complement $\bar{G}$ of a graph $G$ is the graph whose set of vertices is $V(G)$ and whose set of edges is the set of all edges between nodes of $V(G)$ which are not in $E(G)$. 
A subgraph \( H = H(V', E') \) of a graph \( G(V, E) \) is a graph such that \( V' \subseteq V \) and \( E' \subseteq E \) whose elements are incident to vertices in \( V' \).

The degree of a vertex is the number of lines incident with it. A graph is regular if all its vertices have the same degree.

Two graphs \( G \) and \( H \) are isomorphic \((G \cong H)\) if there exists a one to one correspondence \( P \) between their vertices \( x, y \in V(G) \), \((x, y) \in E(G)\) if and only if \((P(x), P(y)) \in E(H)\).

An invariant of a graph \( G \) is a number associated with \( G \) which is the same for any graph isomorphic to \( G \).

A complete set of invariants determines a graph up to isomorphism. No "decent" (Harary, [15]) complete set of invariants is known, but partial sets are used in several heuristic programs for finding graph isomorphism, for example Sussenguth [28] and Unger [31], whose programs are described in the next chapter.

A walk of a graph \( G \) is an alternating sequence of points and lines, beginning and ending with points, in which each line is incident with the two points immediately preceding and following it. It is a path if all points are distinct. If the path is closed, it is a cycle. The length of a path is the
Two points in a graph are **connected** if there exists a path between them. A graph $G$ is **connected** if every two points are connected.

A **directed graph** consists of a finite set $V$ of points together with a collection of pairs of distinct points of $V$. Such a pair $(u,v)$ is called an **arc** (or a **directed line**). Point $u$ is said to be **adjacent to** point $v$, $v$ is **adjacent from** $u$. The **in-degree** of a point $u$ is the number of points adjacent to it, its **out-degree** is the number of points adjacent from it.

A **tree** is a connected graph with no cycles.

A graph is **planar** if it can be drawn in the plane so that no two edges intersect.

A **face** of a planar graph is a region defined by its embedding on the plane. A graph is **labeled** if its points are distinguished by names.

We will use the set $\{1,2,\ldots,n\}$ to label the vertices of graphs. A labeled graph on $n$ points $\{1,\ldots,n\}$ can be represented by its **adjacency matrix** $A=(a_{ij})$, where $A$ is an $nxn$ matrix with
A 1-1 mapping from a finite set onto itself is called a permutation. A permutation $\sigma$ on $n$ elements can be represented by an $n \times n$ permutation matrix $P = (p_{i,j})$ with

$$a_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E(G) \\ 0 & \text{otherwise.} \end{cases}$$

We will refer to a permutation $\sigma$ and its corresponding permutation matrix $P_\sigma$ indifferently as a permutation.

If we label the vertices of two graphs $G$ and $H$ with elements of the same set $\{1,2,\ldots,n\}$ then $G$ and $H$ are isomorphic if and only if there exists a permutation $\sigma$ on the set $\{1,2,\ldots,n\}$ that maps adjacent nodes $x_i, x_j$ of $G$ onto adjacent nodes $y_i, y_j$ of $H$.

An automorphism of a graph $G$ is an isomorphism of $G$ onto itself. The automorphisms of $G$ form a group, $\Gamma(G)$, known as the group of $G$. The automorphism partitioning of $G$ is the collection of orbits of its automorphism group. If the only automorphism of a graph $G$ is the identity, $G$ is an identity graph. Two points $u$ and $v$ of a graph $G$ are similar if $\exists \sigma \in \Gamma(G)$ such that $v = \sigma(u)$. A graph is transitive (or point symmetric) if every two points are similar, that is, if its group acts transitively on its point set.
2.1 Introduction

Graphs occur in a variety of applications (chemistry, electrical engineering, computer science, etc.), and there have been many attempts to find decent algorithms for detecting graph isomorphism. Cook [6] has shown that no polynomial algorithm exists for the subgraph isomorphism problem, i.e. for determining whether a graph G is isomorphic to a subgraph of a graph H. It is still an open question whether a polynomial algorithm exists for the graph isomorphism problem. However, efficient algorithms have been found for certain types of graphs, and Corneil's algorithm is efficient for all but a restricted class of graphs.

There exists of course a simple solution to the problem, namely, if G and H are labeled graphs on n nodes, simply relabel the nodes of G and check for equality with H. Since the number of possible relabelings of G is n!, the number of permutations on n elements, one might have to repeat this step n! times. This will certainly be the case if the graphs are not isomorphic and may be the case even if they are isomorphic, for example if G ≅ H is an identity graph. Thus this method is clearly impractical.

2.2 Heuristic methods

The brute force approach described above tries to match a node in the first graph to each of the nodes of the second
graph. One can try in various ways to limit the set of nodes in H that a node of G can be mapped into, since if G is isomorphic to H, the nodes of G which have some property must map onto nodes of H with the same property. Sussenguth [28] and Unger [31] have proposed algorithms to do this.

Sussenguth proposes an algorithm for matching chemical structures, that is, graphs which have different kinds of vertices and edges, representing respectively the different atoms and the bonds between them. The problem is usually simpler in this case than it would be for unlabeled graphs, since more information is present. The algorithm is as follows: pairs of corresponding sets of nodes from the two graphs are generated by using the following functions on the nodes: node value (i.e., atom type), edge value (type of bond), vertex degree, least number of edges on a cycle from a vertex to itself (if no cycle exists). The sets obtained are checked for agreement of cardinality, since the subsets of nodes having some property must have the same number of elements for the two graphs, if these are isomorphic. Since nodes can be matched only if they have the same value for all the functions considered, if function f yields the set $S \subseteq V(G)$, and corresponding to it the set $S^* \subseteq V(H)$, and function $f_1$ yields the sets $S_1$ and $S_1^*$, then the nodes in the intersection of $S \subseteq V(G)$ and $S_1 \subseteq V(G)$ must correspond to the nodes in the intersection of $S^*$ and $S_1^*$, and the nodes not in the intersection of the sets must also correspond. Therefore the sets obtained by the functions defined on the nodes are intersected and refined as
much as possible. If the cardinality restriction is met, we obtain a partition of the nodes of G into subsets which are mapped onto subsets of nodes of H.

To create new sets, one next looks at the set of nodes adjacent to the nodes of some set S, and matches it to the set of nodes adjacent to the elements of the corresponding set S*. The sets generated are used to refine the partition obtained before. When no more refinements are possible, but some set has more than one element, an arbitrary assignment for some node in the set is made, and again sets are generated by looking at the adjacent nodes, and so on. If a contradiction is reached, the program backtracks and picks another node, if possible, until either an isomorphism is found, or no choices are left, in which case the graphs are not isomorphic.

The program can be slightly modified to check for subgraph isomorphism, by changing the requirement that sets be mapped to sets of equal cardinality to the requirement that they be mapped to sets of equal or greater cardinality.

The algorithm was tested over a thousand times with graphs of 50 points. The time to compute a complete isomorphism averaged about 7 sec. For graphs of about 30 points, this time averaged about 2 sec. Determining that two graphs were not isomorphic was, in 85% of the cases, less than 0.5 msec. No backtracking was ever required. Since the functions used to generate sets are very simple, it is clear that this is due largely to the greater amount of information present in chemical structures, as opposed to graphs.
Unger [31] proposed a much more sophisticated algorithm to test directed graphs for isomorphism which uses the same approach.

Initially, the nodes of the graphs are paired without distinction - any node of G can be mapped to any node of H. The algorithm then evaluates some function on the nodes, and uses the resulting partition of the nodes to refine the partition previously obtained, or to reach a contradiction. The nodal functions used are the following: in-degree, out-degree, number of nodes that can be reached from node i along a path of length \( k, \ k=1,\ldots,n \), a function which is equal to 1 if node i is on a cycle of length \( k, \ k=1,\ldots,n \), and variations of these. The partition is then refined by computing a function on the nodes of a set that takes into account to which sets of the partition the nodes adjacent to them belong, until no further refinement is obtained. If the nodes are partitioned into small enough sets, all possible matchings are tried in order and the graphs compared directly for isomorphism. If the number of possible choices is too large, an arbitrary assignment of a node is made, and the consequences of the assignment explored. If a contradiction is reached, the program backtracks, until no choices are left, in which case the graphs are not isomorphic, or until an isomorphism is found.

The program was reasonably fast in most cases. It ran for 12 minutes without producing a solution when comparing a pair of 16 nodes graphs of the "worst" type. This program uses a large number of functions to limit the number of possible matchings,
II: GRAPH ISOMORPHISM

as well as techniques now proposed by workers in Artificial Intelligence to eliminate unnecessary backtracking (Mackworth, [21]), so that it is unlikely that it can be greatly improved.

2.3 Corneil's algorithm

Corneil [7, 8] has proposed an algorithm for detecting graph isomorphism which is efficient (of the order of $n^5$ at worst) for all graphs which do not contain a transitive 2-strongly regular subgraph. A graph is 2-strongly regular if any two adjacent vertices are both adjacent to a constant number of vertices, and both non adjacent to a constant number of vertices, and similarly for any two non adjacent vertices.

The algorithm derives, for each of the graphs, two unique graphs, the representative graph, which is homomorphic to the given graph, and the reordered graph, which is constructed from the representative graph and is isomorphic to the given graph. It is conjectured that the representative graphs exhibit the automorphism partitioning of the given graphs. For two graphs to be isomorphic, their representative graphs must be identical, since they are uniquely labeled. If the graphs are trees, the converse is also true. It follows from the conjecture that in general if the representative graphs are identical, the given graphs are isomorphic. The reordered graphs give a sufficient condition for isomorphism: if the reordered graphs are identical, the given graphs are isomorphic.
The procedure derives the representative and the reordered graph for each of the given graphs. If the representative graphs are not identical, the graphs are not isomorphic. If the reordered graphs are identical, the graphs are isomorphic and an isomorphism is exhibited. If the representative graphs are identical, but the reordered graphs are different, a counterexample to the conjecture has been found, and some other method has to be used to determine whether the graphs are isomorphic.

The partitioning algorithm used to obtain the representative and the reordered graphs depends directly on the connectivity structure of the graphs, and it is intuitively clear that graphs which are 2-strongly regular graphs, and thus have a high degree of symmetry will present a problem. A modification of the algorithm will work for graphs which contain a transitive h-strongly regular subgraph, h<n, but the computation is then of the order of $n^{e+h}$.

Except for this special case, the algorithm is the most efficient general purpose graph isomorphism algorithm known. In particular, for isomorphic random graphs, the processing time was of the order of $n^e$, and it seems to be efficient in general for graphs encountered in applications.

2.4 Algorithms for special graphs

Planar graphs are a class of graphs for which several efficient isomorphism algorithms have been developed. Trees are
a special case of planar graphs for which a particularly simple algorithm exists, since trees can be put into a canonical form.

A few additional definitions are needed at this point.

A **rooted** tree is one with a distinguished point called its **root**.

A graph is **k-tuply connected** if it contains at least $k+1$ nodes and it is not transformed into an unconnected graph by the deletion of less than $k$ nodes.

An algorithm for ordering the edges of a tree which has been used by various authors (Edmonds [10], Busacker and Saaty [3], Scoins [27] and others) is described in detail by Hopcroft and Tarjan [18].

The algorithm is as follows: first a unique root is found. This is done by eliminating all vertices of degree one from the tree, and repeating this step until either a single vertex (the root) is left, or a single edge is left. In the latter case, a vertex is added in the middle of the edge to create a unique root. Then the vertices at each level are ordered, starting with those farthest away from the root. The leaves at the deepest level are first assigned a value (the same value for all of them). Once the vertices at level $k$ have been assigned values, the vertices at level $k-1$ are ordered lexicographically on the list of values of their sons.

Thus trees can be put into a canonical form, and therefore to compare two trees for isomorphism it is sufficient to compare their canonical forms for identity. This algorithm is linear in the number of nodes in the trees.
Whitney [34] showed that planar triply connected graphs have an embedding in the plane which is unique up to parity, and that planar triply connected graphs are isomorphic if and only if cycles can be mapped to cycles.

Weinberg [32,33] was the first to propose an efficient algorithm for isomorphism of planar triply connected graphs. The algorithm is based on the traversal of an Euler path, that is, a path which traverses every edge exactly once. For each edge of the graph, the graph is traversed starting with this edge, in the clockwise and counter clockwise direction, and a vector code is generated for the traversal. If the graph has b edges, 2b vectors of length 2b+1 are obtained. The same process is repeated with the mirror image embedding of the graph. Thus a 4b x (2b+1) code matrix is generated. The graphs are isomorphic if and only if their code matrices are equal. Since the number of edges in a planar graph satisfies b ≤ 3n-6, where n is the number of vertices, it is clear that the algorithm is polynomial in the number of vertices, and it is in fact of order \( n^2 \).

Hopcroft and Tarjan [17] improved on this result when they presented an algorithm for isomorphism of planar triply connected graphs whose complexity is of order \( n \log n \).

A planar embedding is constructed for the first graph, and the two possible embeddings are constructed for the second graph. For each edge \( e \), treated as two directed edges, a number
\(\lambda(e)\) is computed so that \(\lambda(e_1) = \lambda(e_2)\) iff the number of edges on the face to the right (left) of \(e_1\) is equal to the number of edges on the face to the right (left) of \(e_2\), and the degrees of the heads (tails) of \(e_1\) and \(e_2\) are the same. Edges \(e_1\) and \(e_2\) are said to be distinguishable if there exist edges \(e_3, e_4\), a primary path \(p_1\) from \(e_1\) to \(e_3\) and a corresponding primary path \(p_2\) from \(e_1\) to \(e_4\) such that \(\lambda(e_3) \neq \lambda(e_4)\). Otherwise, \(e_1\) and \(e_2\) are said to be indistinguishable. A primary path is one in which the follower of each edge is the edge to its immediate right or immediate left (in the plane embedding).

It is shown that there exists an isomorphism of \(G\) onto \(H\) mapping \(e_1\) onto \(e_2\) iff \(e_1\) and \(e_2\) are indistinguishable. To find such an isomorphism, therefore, the edges of the graphs are partitioned so that edges are in the same set iff they are indistinguishable. The partitioning algorithm is of the order of \(n \log n\), and the embeddings in the plane can be constructed in time proportional to the number of edges in the graph, therefore linear in the number of nodes of the graph, so that the algorithm is of the order of \(n \log n\).

This algorithm was incorporated by Hopcroft and Tarjan [18] into an algorithm to find an isomorphism between any two planar graphs. In this algorithm, the graphs are divided into connected components, which are divided into biconnected components, and further sub-divided into triply connected components. The connectivity structure of the graph is represented by a tree. The triply connected components are
compared for isomorphism using the algorithm described above, and the connectivity tree reduced, with an isomorphism code replacing the triply connected components. Other substructures of the tree are similarly reduced, until each connected component is reduced to a single vertex with an attached isomorphism code. The codes for the components of each graph are then sorted and compared for equality. The graphs are isomorphic iff the respective codes are equal. Here again the main part of the algorithm is partitioning the triply connected components into equivalence classes, which requires time of the order of $v \log v$, where $v$ is the total number of vertices in the original graphs, and the algorithm is therefore of order $v \log v$.

Hopcroft and Wong [19] have since devised an algorithm for the isomorphism of planar graphs which is linear in the number of vertices of the graphs. From the previous discussion, it is clear that it is enough to restrict attention to triply connected planar graphs. Given a fixed embedding of such a graph, the algorithm associates labels with vertices and edges, and reduces the graph by replacing labeled subgraphs by other labeled subgraphs whose labels encode their origin, until no further reductions are possible. One is then left with one of the five regular polyhedral graphs or with a trivial graph of one vertex, and to test for isomorphism it is sufficient to check for equality of these reduced graphs. The algorithm (which, in the paper mentioned, is not proved to be correct) is
shown to have a running time linear in the number of vertices, but the constant factor is large, so that for small values of n this method is not superior to the previous one. It is hoped that further work in the same direction will yield a more efficient linear algorithm for planar graphs.

2.5 Using the Spectrum of a Graph

Another approach involves the use of algebraic properties of the adjacency matrix of a graph. At a meeting of the American Mathematical Society, Harary (as recalled in Harary, King, Mowshowitz and Read, [16]) hazarded the conjecture that a graph might be characterized by its spectrum, even though it is clear that characteristic polynomials do not in general uniquely characterize matrices, since similar matrices have the same polynomial. This is however not the case, and the conjecture was refuted on the spot. We next present examples of non-isomorphic cospectral graphs.

Collatz and Sinogowitz [5] give a list of the characteristic polynomials of trees with 6, 7 and 8 points, which contains the smallest example of cospectral trees (Fig 2.5.1). Harary et al [16] give an example of a smallest pair of cospectral trees with identical degree partitioning (Fig 2.5.2) as well as examples of smallest cospectral graphs and smallest cospectral connected graphs.
Mowsnowitz [23] presents a method due to A.J. Hoffman for constructing, given any positive integer \( k \), \( k \) non-isomorphic connected regular cospectral graphs, as well as a method for constructing infinitely many pairs of cospectral trees.

Schwenk [26] has shown that, asymptotically, almost all trees have cospectral mates. This behaviour becomes apparent only for trees with a fairly large number of points, which explains why it could have been conjectured that graphs can be
distinguished by their spectra. Schwenk further conjectures that almost every graph has a cospectral mate, even though the asymptotic structure of graphs is very different from that of trees.

Since the characteristic polynomial of a graph is not sufficient to determine isomorphism, Turner [30] has considered generalized matrix functions that could be used. He found, however, that the adjacency matrices of two graphs have the same generalized characteristic polynomial if and only if the graphs have the same number of dissections of certain types, and this is not an invariant of a graph. He exhibits a pair of non-isomorphic generalized cospectral trees on 12 points as well as a pair of non-isomorphic generalized cospectral connected graphs which are not trees.

Mowshowitz (personal communication) has suggested using the additional information given by the eigenvectors of the graphs to determine isomorphism of cospectral graphs. An algorithm for detecting graph isomorphism in this manner is given in chapter 4 of this thesis. A short description of the method follows.

Let $A$ be the adjacency matrix of a graph $G$, $B$ the adjacency matrix of a graph $H$. Then (Chao,4) $G$ and $H$ are isomorphic iff there exists a permutation matrix $P$ such that

\[ P^T A P = B \]
Let \( \lambda_1, \ldots, \lambda_k \) be the distinct eigenvalues of \( A \), \( y \) an eigenvector of \( B \) corresponding to \( \lambda_i \). If \( G \) and \( H \) are isomorphic, we have

\[
P^TAP = B
\]

\[
By_i = \lambda_i y_k \quad \text{for } i = 1, \ldots, k
\]

\[
(P^TAP)y_i = \lambda_i y_i
\]

\[
A(Py_i) = \lambda_i(Py_i)
\]

Thus any eigenvector of \( A \) corresponding to \( \lambda_i \) is a permutation of some eigenvector of \( B \) corresponding to \( \lambda_i \), where the permutation specifies an isomorphism between the graphs.

We next show that the converse is also true.

**Theorem 2.1** Let \( G, H \) be cospectral graphs with \( k \) distinct eigenvalues. Let \( P \) be a permutation matrix such that every eigenvector \( x \) of \( G \) belonging to \( \lambda_i, i=1,\ldots,k \), is the permutation by \( P \) of some eigenvector \( y \) of \( H \) belonging to \( \lambda_i \). Then \( P \) is an isomorphism from \( G \) to \( H \).

**Proof.** Let \( D \) be the diagonal matrix with \( d_{ii} = \lambda_i \). Then

\[
B = QDQ^T,
\]

where \( Q= [y_1, y_2, \ldots, y_n] \), i.e., column \( i \) of \( Q \) is an eigenvector corresponding to \( \lambda_i \).

Similarly, \( A = RDR^T \), where \( R= [x_1, x_2, \ldots, x_n] \)

Since \( x_i = Py_i, \forall i \), \( R = [Py_1, Py_2, \ldots, Py_n] = PQ \). Thus

\[
A = RDR^T = PQD(PQ)^T
\]
that is, \( P \) is an isomorphism between \( G \) and \( H \).

The proposed algorithm uses this relation in the obvious manner: the eigenvectors of two cospectral graphs are compared in order to find a permutation \( P \), if it exists, s.t. each eigenvector of \( A \), when permuted by \( P \), is an eigenvector of \( B \). If no such matrix can be obtained, the graphs are not isomorphic.
3.1 Introduction

The set of all the eigenvalues of the adjacency matrix of a graph is known as the spectrum of the graph. In this chapter, we first review some of the research on the spectra of graphs. This deals mainly with the following problems:

(i) how to find the characteristic polynomial of a graph directly from the graph
(ii) finding classes of graphs which are characterized by their spectra and
(iii) using the spectra of graphs to solve combinatorial problems. We are mainly interested in (i) and (ii), and the relevant results are presented in the next section. We then review results from matrix theory which apply to adjacency matrices of graphs, and numerical methods used to compute the eigenvalues and eigenvectors of a square symmetric matrix.

3.2 Research on the Spectrum of a Graph

The earliest work on this subject is that of Collatz and Sinogowitz [5]. This paper contains a list of the spectra of connected graphs of up to five vertices, and of trees of up to eight vertices. It also includes the first published example of cospectral graphs.

A geometric interpretation for the first few coefficients of the characteristic polynomial of $G$, $\phi(\lambda) = \sum_{i=0}^{n} (-1)^i \lambda^{n-i}$, is given, as well as explicit formulae for the eigenvalues of some special
graphs, and their eigenvectors.

Let $G$ be the complete graph. Then $\phi(\lambda) = (n-1-\lambda)(-\lambda-1)^{n-1}$; for $\lambda = n-1$, the corresponding eigenvector is $x_i = (1, \ldots, 1)$; for $\lambda = -1$, $i=2, \ldots, n$ $x_i$ is arbitrary as long as $\sum_{i=1}^{n} x_i = 0$.

Let $G$ be a path. Then $\lambda = 2\cos \omega$, where $\omega = \frac{i\pi}{n}$, $i=1, \ldots, n$; the corresponding eigenvector is $x^{(i)} = \sin i \omega$, $i=1, \ldots, n$.

Let $G$ be a cycle. Then $\lambda = 2 \cos \xi$, where $\xi = \frac{2\pi i}{n}$, for $i=1, \ldots, n$;

the corresponding eigenvectors are $x^{(i)} = \cos (k \xi)$, $x^{(i)} = \sin (k \xi)$.

The eigenvalues of a path are simple, the eigenvalues of a cycle are double, except for $i=n$ and $i=n/2$ if $n$ is even.

Collatz and Sinogowitz also show that

$$q_{\min} \leq \tilde{q} \leq \lambda \leq q_{\max},$$

where $\lambda$ is the index of the graph (i.e., its maximal eigenvalue), $q_{\min}$, $\tilde{q}$, $q_{\max}$ respectively the minimum, average and maximal degree of a vertex. Equality holds if the graph is regular.

Mowshowitz [23] generalizes these observations, giving a general formula for the coefficients of the characteristic polynomial of a graph in terms of its subgraphs.

For an arbitrary digraph $H$ of order $k$, let $f_\mu([i, i, \ldots, i])$ denote the number of collections of disjoint directed cycles in $H$ of lengths $i, i, \ldots, i$, where $i \geq 1$ ($1 \leq j \leq r$) and $i_1 + i_2 + \ldots + i_r = k$.

Let $D$ be a digraph of order $n$. Then for $1 \leq k \leq n$, the $k$-th
coefficient $a_k$ of the characteristic polynomial of $A(D)$ is given by

$$a_k = \sum \frac{1}{k!} (-1)^{i_k} f_D(\{i_1, i_2, \ldots, i_r\}),$$

where the summation is taken over all rank $r$ partitions $\{i_1, i_2, \ldots, i_r\}$ ($1 \leq r \leq k$) of $k$, and $a_0 = 1$.

In an undirected graph, if for a given partition $(i_1, i_2, \ldots, i_r)$ of $k$ we let

$$g_i = \begin{cases} 1 & \text{if } 1 \leq i < 2 \\ 2 & \text{if } i \geq 2, \end{cases}$$

and define $f_G(\{i_1, i_2, \ldots, i_r\})$ as above but for undirected cycles (and lines), we obtain:

Let $G$ be a graph of order $n$. Then for $1 \leq k \leq n$ the $k$-th coefficient $a_k$ of the characteristic polynomial of $A(G)$ is given by

$$a_k = \sum \frac{1}{k!} (-1)^{i_k} g(i_k) f_G(\{i_1, i_2, \ldots, i_r\})$$

where the summation extends over all rank $r$ partitions $\{i_1, i_2, \ldots, i_r\}$ ($1 \leq r \leq k$) of $k$, and $a_0 = 1$.

In particular, for an undirected graph without loops, $a_0 = 1$, $a_1 = 0$ and $a$ is equal to the number of edges.

The coefficients of the characteristic polynomial can also be computed using a recurrence formula, and an explicit formula is given for paths, trees homeomorphic to stars, and line stars. A list of all trees of up to 10 points with the coefficients of their characteristic polynomial is given, which corrects some errors in the list given in [5].

Harary, King, Mowshowitz and Read [16] give explicit expressions for the characteristic polynomial of the $n$-cube $Q_n$. 
\[ \phi(Q_n) = \left\{ \begin{array}{ll}
1 & \text{if } r = n - 2k \\
(n-2k)^{\frac{r}{n}} & \text{otherwise}
\end{array} \right. \]
and the complete \( t \)-partite graph \( K(n_1,n_2,\ldots,n_t) \)
\[ \phi(K(n_1,n_2,\ldots,n_t)) = \sum_{r=1}^{t} (1-r) s_r(n_1,n_2,\ldots,n_t) \lambda^{-t} \]
where \( s_r(n_1,n_2,\ldots,n_t) \) denotes the sum of the products \( r \) at a time of the numbers \( n_1,n_2,\ldots,n_t \) and \( S_1 = 1 \).

As a corollary we obtain the characteristic polynomial of the complete bipartite graph \( K_{m,n} \)
\[ \phi(K_{m,n}) = \lambda^{m-1} \left( \lambda^2 - mn \right) \]
and of the star \( K_{1,n} \)
\[ \phi(K_{1,n}) = \lambda^n (\lambda^2 - n). \]

Turner [29] has considered graphs with a prime number of points which are point symmetric. A starred polygon is a graph \( G=(V,E) \) whose vertices may be labeled in such a way that \((v_i,v_{i+k}) \in E \) iff \((v_{i+k},v_i) \in E\), for \( k=1,\ldots,|V|-1 \), where the subscripts are taken modulo \(|V|\).

Turner shows that a connected graph with a prime number of points \( p \) is point symmetric iff it is a starred polygon, and he further shows that these graphs are characterized by their spectrum, that is, two point symmetric graphs with a prime number of points are isomorphic iff they have the same eigenvalues.

If the vertices of a starred polygon are labeled \( v \) to \( v \) in the clockwise direction, the adjacency matrix of the graph is a circulant, that is, a matrix in which each row after the first one is obtained from the preceding one by shifting it cyclically one position to the right.
Elpas and Turner [11] generalize this result, showing that if two circulant matrices $A$ and $B$ of prime order $p$, $p>2$, with rational entries have the same eigenvalues, then there exists a permutation matrix $P$ s.t. $B=P^TAP$.

It follows immediately that two directed graphs having a prime number of vertices and circulant adjacency matrices are isomorphic iff they have the same eigenvalues.

The next results are quoted from Cvetkovic [9], who presents a comprehensive survey of the spectra of graphs and their use in different combinatorial problems.

If a graph is regular of degree $r$, to each eigenvalue $\lambda_i(\lambda_i \neq r)$ with multiplicity $p$, there corresponds in the spectrum of $G$ the eigenvalue $-\lambda_i - 1$, whose multiplicity is also $p$.

It is clear that two graphs $G$ and $H$ are isomorphic iff their complements are isomorphic. Thus one might be tempted, if the eigenvalues of $G$ have high multiplicity, to look at the spectrum of $G$ instead. The following theorem shows that this will not make any significant difference.

**Theorem 3.2.1.** If the spectrum of $G$ contains eigenvalue $\lambda_i$ with multiplicity $p>1$ then the spectrum of the complement $\overline{G}$ contains the eigenvalue $-\lambda_i - 1$ with multiplicity $\overline{p}$ s.t. $p-1<\overline{p}<p+1$.

The spectrum characterizes all those graphs determined by the number of vertices and the number of edges, since these numbers are determined by the spectra. Thus graphs without edges, graphs with one edge and their complements are
characterized by their spectra. Regular graphs of degree 1 and n-2 are determined by their spectra, since there exists only one regular graph of degree 1 on n points, and the spectra of regular graphs and their complement are mutually determined.

Finck [13] has shown that regular graphs of degree 2 (and therefore also n-3) are characterized by their spectra.

The line graph of a graph G, L(G), is a graph whose points are the lines of G, with two points in L(G) adjacent whenever the corresponding lines of G are. Line graphs of some special graphs, as well as graphs which are direct sums of complete graphs are characterized by their spectra. Sachs [25] showed that the characteristic polynomial of the line graph L(G) of a regular graph G of degree r satisfies

\[ f_{L(G)}(\lambda) = (\lambda+2)^{m-2} f_G(\lambda-r+2), \]

where m is the number of vertices of G.

**Theorem 3.2.2.** A graph containing at least one edge is bipartite if its spectrum, taken as a set of points on the number axis, is symmetric with respect to the zero point.

**Theorem 3.2.3.** A connected graph, containing at least one edge, with index 2, is bipartite iff -2 is in its spectrum.

It is easily seen by direct substitution that the eigenvectors of a bipartite graph corresponding to -\( \lambda \); can be obtained from the eigenvector corresponding to \( \lambda \) by multiplying the components corresponding to one set of edges by -1.
**Theorem 3.2.4.** If $G$ is a connected graph with diameter $D$, and $G$ has $m$ distinct eigenvalues, then $m \geq D + 1$.

**Theorem 3.2.5.** Let $\{\lambda_1, \ldots, \lambda_n\}$ be the spectrum of $G$, where $r$ is the index of $G$. $G$ is regular iff

$$\frac{1}{n} \sum_{i=1}^{n} \lambda_i^1 = r \quad (\ast)$$

If $(\ast)$ holds and $r$ has multiplicity $p$, the graph has $p$ components.

**Theorem 3.2.6.** If the index of $G$ has multiplicity $p$ and a positive eigenvector corresponding to it, $G$ has $p$ components.

Some of the above results are proved from structural properties of the graphs, others follow from properties of the adjacency matrix.

### 3.3 Algebraic properties

The adjacency matrix of an undirected graph is a symmetric matrix of 0's and 1's, with a zero diagonal. Such matrices have several important properties. We list here several results that are relevant to our algorithm. For proofs, see for example [2](theorems 1-4) and [14](theorems 5-8).

**Theorem 3.3.1.** The eigenvalues and eigenvectors of a real symmetric matrix are all real.
THEOREM 3.3.2. The eigenvectors associated with distinct eigenvalues of a real symmetric matrix are mutually orthogonal.

THEOREM 3.3.3. If \( \lambda \) is an eigenvalue of multiplicity \( k \) of a real symmetric matrix, then associated with \( \lambda \) is an eigenvector space of dimension \( k \).

THEOREM 3.3.4. The sum of the eigenvalues of a matrix \( A \) is equal to the trace of \( A \), that is, \( \sum \lambda_i = \sum a_{ii} \). In particular, if \( A \) is the matrix of a graph without loops, \( \sum \lambda_i = 0 \).

A matrix \( A \) is called non-negative if all its elements are non-negative.

A square matrix \( A \) is called reducible if there is a permutation that puts it in the form \( \tilde{A} = \begin{bmatrix} B & * \\ 0 & D \end{bmatrix} \), where \( B, D \) are square matrices. Otherwise \( A \) is called irreducible.

It is easy to see that the adjacency matrix of a graph \( G \) is irreducible iff \( G \) is connected. Since these are the graphs we are mainly interested in the following theorem is of great importance.

THEOREM 3.3.5. (Frobenius) An irreducible non-negative matrix \( A \) always has a positive eigenvalue \( r \) that is a simple root of the characteristic equation. The moduli of all the other eigenvalues does not exceed \( r \). To the maximal eigenvalue \( r \)
there corresponds an eigenvector with positive coordinates (often referred to as the Perron vector). Moreover, if $A$ has $h$ eigenvalues of modulus $r$, then these are all distinct, and are roots of the equation $\lambda - r = 0$.

**Theorem 3.3.6.** Let $s_i = \sum_{j=1}^{n} a_{ij}$ for $i = 1, 2, \ldots, n$, $s = \min_{i \leq n} s_i$, $S = \max_{i \leq n} s_i$. For an irreducible non-negative matrix $A$, $s \leq r \leq S$, and equality holds only if $s = S$, i.e., only when all the row sums are equal.

In particular, if $A$ is the adjacency matrix of a graph $G$, $r \leq n-1$, and for all $i$, $-(n-1) < \lambda < n-1$. If $G$ is regular of degree $k$, $r = k$, and for all $i$, $-k \leq \lambda \leq k$.

**Theorem 3.3.7.** An irreducible non-negative matrix $A$ cannot have two linearly independent non-negative eigenvectors.

A matrix is **stochastic** if all its row sums are equal to 1.

**Theorem 3.3.8.** A non-negative matrix $A$ is stochastic iff its maximal eigenvalue is 1, and it has the eigenvector $(1, 1, \ldots, 1)$ corresponding to the eigenvalue 1.

**Corollary 3.3.8A**

If $A$ is the adjacency matrix of a regular graph of degree $k$, its maximal eigenvalue is $k$ and an eigenvector corresponding to it is $(1, 1, \ldots, 1)$. 

III: THE SPECTRUM OF GRAPHS
Symmetric matrices then have nice properties: the eigenvalues and eigenvectors are all real, the eigenvector space is n-dimensional, eigenvectors associated with distinct eigenvalues are orthogonal, etc. This ensures the existence of stable and efficient numerical algorithms for computing their eigenvalues and eigenvectors. Since we are concerned with matrices which are adjacency matrices of graphs, we could find the characteristic polynomial using one of the formulae of the preceding section, but as these in general involve looking at all the subgraphs of a graph, they are not efficient.

3.4 Computation of eigenvalues and eigenvectors

Eigenvalues and eigenvectors of square matrices are usually computed using iterative methods, rather than by solving the characteristic equation. Several such methods exist (see for example Wilkinson, [35]), and libraries usually provide standard routines for finding eigenvalues and eigenvectors of symmetric matrices. The routine we used first transforms the given matrix to a similar tridiagonal matrix, using Householder's algorithm, and then finds the eigenvalues and eigenvectors (of the original matrix) using the QR algorithm. A full description of the algorithm, as well as ALGOL programs implementing it, can be found in Martin, et al [21] and Bowdler, et al [1]. All the computations were done in double precision (on an IBM 370/168), and the results were always satisfactory. It is possible that one might run into problems with matrices having pathologically
close eigenvalues, but this was never the case with the examples we tried.

If $G$ is not connected, its adjacency matrix is reducible, and can be put in the form $A = \begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix}$. It is clear that the eigenvalues of $A$ consist of the eigenvalues of $B$ and those of $C$. If $x=(x_1, x_2, \ldots, x_n)$ is an eigenvector of $B$ corresponding to $\lambda$, then $u=(x_1, x_2, \ldots, x_n, 0, \ldots, 0)$ is an eigenvector of $A$ corresponding to $\lambda$. Similarly, if $y=(y_1, y_2, \ldots, y_n)$ is an eigenvector of $C$ corresponding to $\mu$, then $v=(0, \ldots, 0, y_1, \ldots, y_n)$ is an eigenvector of $A$ corresponding to $\mu$.

The computation time for finding the eigenvalues and eigenvectors of a symmetric matrix is of order $n^3$, where $n$ is the dimension of the matrix. Thus it is clearly inefficient to compute the eigenvalues of a reducible matrix directly from the matrix. The computation time and the error will be much smaller if the spectrum is computed separately for the non-zero blocks $B$ and $C$. That is, when computing the eigenvalues of a non-connected graph, it is best to compute separately the eigenvalues of each connected component. This of course is the case with most algorithms for graph isomorphism: if the computation time is a non-linear function of the number of vertices, it is worthwhile, if possible, to decompose the problem into one of matching a number of smaller graphs for isomorphism. Thus, although our algorithm can be used for connected or disconnected graphs, it is best when comparing disconnected graphs to test their connected components for isomorphism.
4.1 Intuitive Description of the Algorithm

We have seen in Section 2.5 that if each eigenvector of a graph $G$ is a permutation of some eigenvector of a cospectral graph $H$ corresponding to the same eigenvalue, then $G$ is isomorphic to $H$. We want to look at the eigenvectors of $G$ and $H$ and use them to find such a permutation.

For example, consider the graphs in Fig 4.1.1. The eigenvalues and eigenvectors of the graphs were computed using a standard library routine. The graphs are cospectral, and we want to find out whether they are isomorphic or not. First note that all the eigenvalues in this case are simple. The routine used to compute the spectrum yields orthonormalized vectors, and we therefore know that if the graphs are isomorphic there exists a permutation matrix $P$ s.t.

$$x_i = Py_i \text{ or } x_i = -Py_i, \quad i = 1, \ldots, n$$

(if the vectors were not normalized, we would have to find constants $c_i$, not necessarily $+1$ or $-1$, s.t. $x_i = c_iPy_i$. This can easily be done, for example by comparing the largest components of $x_i$ and $y_i$).

We need the following definitions at this point:

Let $S = \{s_1, \ldots, s_m\}$, $T = \{t_1, \ldots, t_n\}$ be partitions of $\{1, \ldots, n\}$.

A partial permutation $P^*$ is a 1-1 mapping from $S$ to $T$ s.t. elements are mapped to elements of equal cardinality.
IV: THE EIGENVector CONNECTION

EIGENVALUES

-1.8513 -1.3504 -0.5600 0.0000 1.0561 2.7056

EIGENVECTORS

0.2619 -0.2355 -0.6754 0.0 -0.6414 -0.0920
0.0662 0.7137 -0.3959 0.0 0.2555 -0.5140
-0.3792 -0.3849 -0.1210 0.7071 0.1719 -0.4048
0.6358 -0.1939 0.4636 0.0 -0.0740 -0.5812
-0.3792 -0.3849 -0.1210 -0.7071 0.1719 -0.4048
-0.4849 0.3180 0.3782 0.0 -0.6774 -0.2488

Figure 4.1.1 Two Cospectral Isomorphic Graphs
IV: THE EIGENVECTOR CONNECTION

We will informally refer to a mapping from a partition of the nodes of a graph $G$ (labeled 1, ..., $n$) to a partition of the nodes of a graph $H$ (similarly labeled) as a partial permutation from $G$ to $H$. We will refer to a partial permutation whose domain consists of only singletons as a permutation.

Let $S, S', T, T'$ be partitions of $\{1, \ldots, n\}$, $P^*: S \rightarrow T$, $Q^*: S' \rightarrow T'$ be partial permutations. If for each $s'$ in $S'$ there is an $s$ in $S$ such that $s' s$ and $Q^*(s') \subseteq P^*(s)$, $Q^*$ is said to refine $P^*$, and $P^*$ is said to contain $Q^*$.

A partial permutation $P^*$ is maximal with respect to some property if it is not contained in any partial permutation $Q^*$ with the same property. A partial permutation $P^*$ from the nodes of a graph $G$ to the nodes of a graph $H$ is said to be proper if every permutation $P$ refining it is an isomorphism from $G$ to $H$.

By Theorem 2.1, a partial permutation $P^*$ s.t. every permutation $P$ refining it maps each eigenvector of $G$ onto an eigenvector of $H$ corresponding to the same eigenvalue is a proper partial permutation.

We now return to the graphs of Fig 4.1.1.

Comparing the components of $x$, and $y$, we see that for each $1 < i < 6$, there exists $j, 1 < j < 6$, s.t. $x^{(i)} = -y^{(k)}$, $j \neq k$ for $i \neq k$. Thus it is possible to find a matching between $x$ and $y$. This matching yields a partial permutation from the nodes of $G$ onto the nodes of $H$. 
Considering now the remaining eigenvectors, we see that none of them gives additional insight as to how to match the two nodes (3,5) of G to the two nodes (4,6) of H, while all of them satisfy

\[ x_i = Py_i \text{ or } x_i = -Py_i, \quad i = 2, \ldots, 6, \] 

P refining \( P^* \)

Therefore, by Theorem 2.1, any partition refining \( P^* \) specifies an isomorphism.

The graphs in Fig 4.1.2 form another pair of cospectral graphs. These are the smallest cospectral non-isomorphic trees, already mentioned in Section 2.5.

Comparing \( x \) and \( y \), it is immediately clear that no matching between their components is possible, since the components of \( x \) have 4 distinct values (and only 2 distinct absolute values), while the components of \( y \) have 5 distinct values (5 distinct absolute values). Thus there can exist no \( P \) s.t.

\[ x_i = Py_i \text{ or } x_i = -Py_i, \]

therefore no \( P \) s.t.

\[ P^T A P = B, \]

and the graphs are not isomorphic.

Consider now the graph in Fig 4.1.3, which we want to map onto itself.
### IV: THE EIGENVECTOR CONNECTION

#### Eigenvalues

-2.3028 -1.3028 1.3028 2.3028 -0.0000 -0.0000 -0.0000 -0.0000

#### Eigenvectors

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2454</td>
<td>-0.3263</td>
<td>-0.3263</td>
<td>0.2454</td>
<td>0.0</td>
<td>0.7887</td>
<td>-0.2113</td>
<td>0.0</td>
</tr>
<tr>
<td>0.2454</td>
<td>-0.3263</td>
<td>-0.3263</td>
<td>0.2454</td>
<td>0.0</td>
<td>-0.2113</td>
<td>0.7887</td>
<td>0.0</td>
</tr>
<tr>
<td>0.2454</td>
<td>-0.3263</td>
<td>-0.3263</td>
<td>0.2454</td>
<td>0.0</td>
<td>-0.5774</td>
<td>-0.5774</td>
<td>0.0</td>
</tr>
<tr>
<td>-0.5651</td>
<td>0.4250</td>
<td>-0.4250</td>
<td>0.5651</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.5651</td>
<td>0.4250</td>
<td>0.4250</td>
<td>0.5651</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>-0.2454</td>
<td>-0.3263</td>
<td>0.3263</td>
<td>0.2454</td>
<td>-0.8157</td>
<td>0.0</td>
<td>0.0</td>
<td>0.367</td>
</tr>
<tr>
<td>-0.2454</td>
<td>-0.3263</td>
<td>0.3263</td>
<td>0.2454</td>
<td>0.4396</td>
<td>0.0</td>
<td>0.0</td>
<td>0.6880</td>
</tr>
<tr>
<td>-0.2454</td>
<td>-0.3263</td>
<td>0.3263</td>
<td>0.2454</td>
<td>0.3760</td>
<td>0.0</td>
<td>0.0</td>
<td>-0.7247</td>
</tr>
</tbody>
</table>

Figure 4.1.2 Smallest Cospectral Non-Isomorphic Trees
### IV: The Eigenvector Connection

#### Eigenvalues

-2.4560  -1.6180  -0.9229  -0.0000  0.6180  1.0650  3.3139

#### Eigenvectors

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.3829</td>
<td>0.0</td>
<td>0.7350</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2820</td>
<td>0.4833</td>
</tr>
<tr>
<td>-0.0355</td>
<td>-0.6015</td>
<td>-0.4289</td>
<td>0.0</td>
<td>-0.3717</td>
<td>0.4057</td>
<td>0.3875</td>
</tr>
<tr>
<td>0.5058</td>
<td>0.3717</td>
<td>0.0897</td>
<td>0.0</td>
<td>-0.6015</td>
<td>-0.2556</td>
<td>0.4133</td>
</tr>
<tr>
<td>-0.4119</td>
<td>0.0</td>
<td>-0.1945</td>
<td>-0.7071</td>
<td>0.0</td>
<td>-0.4799</td>
<td>0.2494</td>
</tr>
<tr>
<td>0.5058</td>
<td>-0.3717</td>
<td>0.0897</td>
<td>0.0</td>
<td>0.6015</td>
<td>-0.2556</td>
<td>0.4133</td>
</tr>
<tr>
<td>-0.0355</td>
<td>0.6015</td>
<td>-0.4289</td>
<td>0.0</td>
<td>0.3717</td>
<td>0.4057</td>
<td>0.3875</td>
</tr>
<tr>
<td>-0.4119</td>
<td>0.0</td>
<td>-0.1945</td>
<td>-0.7071</td>
<td>0.0</td>
<td>-0.4799</td>
<td>0.2494</td>
</tr>
</tbody>
</table>

---

**Figure 4.1.3** A 7-point Graph
Comparing $x$ and $y$, we see that there is one maximal partial permutation that maps the elements of $x$ onto the elements of $y$

$$P^* : 1 \rightarrow 1, (2,6) \rightarrow (2,6), (3,5) \rightarrow (3,5), (4,7) \rightarrow (4,7)$$

If we now try to match $x^*$ and $y^*$, we obtain 2 partial permutations,

$$R^* : (1,4,7) \rightarrow (1,4,7), 2 \rightarrow 2, 6 \rightarrow 6, 3 \rightarrow 3, 5 \rightarrow 5$$
$$R^* : (1,4,7) \rightarrow (1,4,7), 2 \rightarrow 6, 6 \rightarrow 2, 3 \rightarrow 5, 5 \rightarrow 3$$

Since $x^* = x^{\prime\prime} = y^*$, we have

$R^* \text{ satisfies: } \forall R \text{ refining it, } x^* = Ry^*$

$R^* \text{ satisfies: } \forall R \text{ refining it, } x^* = -Ry^*$

We will refer to a permutation of the first kind as positive with respect to $x$, and a permutation of the second kind as negative with respect to $x$.

To obtain a partial permutation which will map $x$ onto $y$, and $x^*$ onto $y^*$, we refine $P^*$ using $R^*$ and $R^*$, yielding $P^*$ and $P^*$

$$P^* : 1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3, (4,7) \rightarrow (4,7), 5 \rightarrow 5, 6 \rightarrow 6$$
$$P^* : 1 \rightarrow 1, 2 \rightarrow 6, 3 \rightarrow 5, (4,7) \rightarrow (4,7), 5 \rightarrow 3, 6 \rightarrow 2$$

Comparing $x^*$ and $y$, we see that any refinement of $P^*$ and $P^*$ will map $x$ onto $y$.

Comparing $x^*$ and $y^*$, we obtain 2 partial permutations, one positive and one negative, but these are of no use in refining $P^* \text{ and } P^*$, since this only means that the set $(4,7)$ can be broken up in two ways, which is clear.

Comparing $x^*$ and $y^*$, we obtain 2 partial permutations, one positive and one negative, which are equal to $P^* \text{ and } P^*$ respectively.

Comparing $x^*$ and $y^*$, we obtain a partial permutation which
IV: THE EIGENVector CONNECTION

contains $P_1^*$ and $P_2^*$.

Comparing $x^*\text{and } y^*$, we obtain again a partial permutation
which contains $P_1^*$ and $P_2^*$, and thus cannot be used to refine
them further.

It is clear that the two partial permutations which we
obtained, $P_1^*$ and $P_2^*$, are the only maximal proper partial
permutations mapping the graph onto itself, and therefore the
set of permutations refining them comprises the automorphism
group of the graph.

In the above examples, the eigenvectors compared belonged
to eigenvalues of multiplicity one. The comparison in this case
is very simple. The situation is considerably more complicated
when we look at eigenvectors belonging to eigenvalues of
multiplicity greater than one. In this case, we no longer have
only two possible mappings between corresponding eigenvectors of
G and H, and it is therefore hopeless to try and find all the
isomorphisms from G to H by this method. We try to find one
isomorphism by finding one mapping between corresponding
eigenvectors, and backing up if our choice was wrong.

Let $\lambda_1$ be an eigenvalue of G and H of multiplicity 2. If G
and H are isomorphic, we know that $x$ is the permutation of some
eigenvector of H belonging to $\lambda_1$. Thus we only know that there
exists $a_1$, $a_2$, and a permutation matrix $P$ s.t. $x = a_1Py + a_2Py$.
In order to find $a_1$ and $a_2$, we solve a set of 2 linear equations
of the form
The k vectors that span the vector space associated with an eigenvalue of multiplicity k form a matrix \([v_1, \ldots, v_k]\) of column-rank k, since they are linearly independent, and therefore also of row-rank k. Thus a system such as (1) which is linearly independent can always be found by trying all possible values for \(l\) and \(m\). If \(G\) and \(H\) are isomorphic, then by varying \(l\) and \(m\) over all combinations of \([1, \ldots, n]\) and varying \(j\) and \(k\) over all combinations of \([1, \ldots, n]\) and over all permutations of these combinations, we will find all solutions s.t. both (1) and (2) hold, yielding candidates \(P\) for the isomorphism.

For example, consider the graphs in Fig 4.1.4. Comparing the eigenvectors associated with \(\lambda\), we find that \(x_1 = -Py_1\), where

\[
P\ast = (1,2,3) \rightarrow (1,2,5), (4,6,8) \rightarrow (3,4,6) (5,7) \rightarrow (7,8)
\]

for every permutation \(P\) refining \(P\ast\).

We next try to solve a linear system of one equation, where the elements of \(x_1\) and \(y_1\) are chosen according to \(P\ast\). Choosing \(x_1^{(1)}\) and \(y_1^{(1)}\), and solving \(x_1 = ay_1\), and then comparing \(x_1\) to \(ay_1\), we obtain
EIGENVALUES
-2.5616 -0.0000 1.5616 3.0000 -1.6180 -1.6180 0.6180 0.6180

EIGENVECTORS
-0.0903 0.0 0.4474 -0.3536 -0.6841 0.1199 0.0003 0.4293
-0.0903 0.0 0.4474 -0.3536 0.2382 -0.6524 -0.3719 -0.2144
-0.0903 0.0 0.4474 -0.3536 0.4459 0.5325 0.3716 -0.2149
0.4121 0.0 -0.1962 -0.3536 -0.2756 -0.3291 0.6013 -0.3477
-0.4827 -0.7071 -0.3769 -0.3536 0.0 0.0 0.0 0.0
0.4121 0.0 -0.1962 -0.3536 0.1472 0.4032 -0.6017 -0.3469
-0.4827 0.7071 -0.3769 -0.3536 0.0 0.0 0.0 0.0
0.4121 0.0 -0.1962 -0.3536 0.4228 -0.2741 0.0004 0.6946

Figure 4.1.4 Two Cospectral Isomorphic Graphs
This partial permutation will also map $x_j$ to $y_s$, and $x_k$ to $y_s$. We next try to map $x_s$ to a linear combination of $y_s$ and $y_t$. In order to find the constants, we try to solve the equations
\[ x_s^{(s)} = a_1 y_s^{(s)} + a_2 y_t^{(s)} \]
\[ x_s^{(t)} = a_1 y_s^{(t)} + a_2 y_t^{(t)} \]
since these have to correspond if our partial permutation contains a proper permutation. The linear system obtained, however, is not linearly independent, and we therefore have to try another choice. Solving
\[ x_s^{(s)} = a_1 y_s^{(s)} + a_2 y_t^{(s)} \]
\[ x_s^{(t)} = a_1 y_s^{(t)} + a_2 y_t^{(t)} \]
and comparing $x_s$ to $a_1 y_s + a_2 y_t$, we obtain
\[ P_1^*:(1 \ 2 \ 3)\rightarrow(1 \ 2 \ 5), \ (4 \ 6 \ 8)\rightarrow(3 \ 4 \ 6), \ 5\rightarrow8, \ 7\rightarrow7 \]
We now have a partial permutation from $G$ to $H$ which maps the five first eigenvectors of $G$ onto the corresponding eigenvectors of $H$. Since $P_1^*$ is in fact a permutation, it cannot be refined any further. At this point, we can either check the graphs directly to see whether it is an isomorphism, or compare the next three eigenvectors to ensure that $P_1^*$ is a proper partial permutation. Either way, we will find it is indeed an isomorphism from $G$ to $H$.

4.2 Connection with the Group of the Graph

We next look at the relation between our algorithm and the group of the graph.
Lemma 4.2.1. Let $G$ and $H$ be isomorphic graphs. Let $P*$ be a maximal proper partial permutation from the nodes of $G$ to the nodes of $H$, $S=\{s_1, \ldots, s_k\}$ its domain, $T=\{t_1, \ldots, t_k\}$ its range, $P*(s_i)=t_i$, \forall i. Denote by $S_{s_i}$ the symmetric group on $s_i=\{a_1, \ldots, a_{|s_i|}\}$ (that is, the group consisting of all permutations of the elements of the set $s_i$, regarded as permutation matrices). Then the group $\Gamma' = \langle p* \rangle = \langle s_{t_1}, S_{s_1}, \ldots, S_{s_k} \rangle$ is a subgroup of $\Gamma(G)$.

Proof. $\Gamma'$ is clearly a group. We show each element of $\Gamma'$ is an automorphism of $G$.

Let $\gamma \in \Gamma'$. There exist $r_1, \ldots, r_k$ such that $\gamma = r_1 \cdots r_k, r_i \in S_{t_i}$. Let $p_i$ be a permutation refining $p$. Then $p_i = s_{t_i} = r_1 \cdots r_k p_i$ is also a refinement of $p*$, since $r_i$ only permutes elements of $s_i$, for all $i$. Since $p*$ is a proper partial permutation, we have

\[
\begin{align*}
\gamma p_i A p_i &= B \\
p_i A \gamma p_i &= B \\
(\gamma p_i) A (\gamma p_i) &= B \\
\gamma A \gamma &= p_i BP_i = A
\end{align*}
\]

$\gamma \in \Gamma(G)$.

We next show how the group of a graph can be constructed if all the maximal proper partial permutations from it to an isomorphic graph (or to itself) are known.

Theorem 4.2.2. Let $G, H$ be isomorphic graphs. Let $P_1, \ldots, P_k$
be all the maximal proper partial permutations from $G$ to $H$. Let $< P_i^* >$ denote the group of permutations induced by $P_i^*$, as in Lemma 4.2.1. Then $\Gamma(G) = < < P_i^* >, P_i^*P_j^* >$, where $P_m$ is some permutation refining $P_m^*$.

**Proof** (i) By Lemma 4.2.1, if $e < P_i^* >, \sigma \in \Gamma(G)$. If $P_1, P_2$ are isomorphisms from $G$ to $H$, then

$$P_1^T A P_1 = P_2^T A P_2,$$

$$P_2 P_1^T A P_1 P_2^T = A,$$

therefore $P_1 P_2^T \in \Gamma(G)$.

(ii) Let $x \in \Gamma(G), Q$ an isomorphism from $G$ to $H$. Then there exists an $R$ s.t.

$$xQ = R$$

$$x = RQ^T.$$ 

Since all isomorphisms are refinements of $P_i^*$'s (because of the maximality of the $P_i^*$'s), we have

$$Q = q_i... q_1 P_i, R = r_i... r_1 P_i \text{ where } e \in < P_i^* >, e \leq < P_i^* >$$

$$x = RQ^T = q_i... q_1 P_i P_i^T... P_i^T \in < < P_i^* >, P_i^*P_j^* >.$$

For example, we show how to obtain the group of the graph in Fig 4.2.1.

Matching $x_1$ and $y_1$, we obtain two maximal partial permutations

$$P*: (1,3) \rightarrow (1,3), (2,4) \rightarrow (2,4)$$

$$Q*: (1,3) \rightarrow (2,4), (2,4) \rightarrow (1,3)$$

Comparing $x_2$ and $y_2$ does not yield a refinement of $P^*$ or
Q*. Clearly, for every permutation P refining P* or Q*, and for every x belonging to $\lambda_3$, there is a y belonging to $\lambda$, s.t. $x = Py$, so that P* and Q* are maximal proper partial permutations. Taking $P = (1) , (2) , (3) , (4)$ , $Q = (1 \ 2 \ 3 \ 4)$ (the cyclic permutation on four elements), we have, by Theorem 4.2.2, $\Gamma(G) = \langle (1 \ 3) , (2 \ 4) , (1 \ 2 \ 3 \ 4) \rangle$.

**Theorem 4.2.3.** Let $P*: S \rightarrow T$, $Q*: S' \rightarrow T'$ be two distinct maximal proper partial permutations from a graph G to an isomorphic graph H. Then $S = S'$.

**Proof** Let $P*, Q*$ be maximal partial permutations from G to H. Without loss of generality, let us assume

- $P*: 1 \rightarrow 1$, $2 \rightarrow 2$, ...
- $Q*: (1, 2) \rightarrow (3, 4)$, ...

Let P be a permutation refining $P*$ and $Q_1, Q_2$ be permutations refining $Q*$. s.t.

- $Q_1(1) = 3$, $Q_1(2) = 4$
- $Q_2(1) = 4$, $Q_2(2) = 3$, $Q_1 \upharpoonright_{\{3, 4\}} = Q_2 \upharpoonright_{\{3, 4\}}$

$Q_2^T Q_1$ is an automorphism of H. Now let $P' = P Q_2^T Q_1$,

- $P'(1) = P Q_2^T Q_1(1) = 2$,
- $P'(2) = P Q_2^T Q_1(2) = 1$, and
- $P' \upharpoonright_{\{3, 4\}} = P \upharpoonright_{\{3, 4\}}$. 

IV: THE EIGENVECTOR CONNECTION

EIGENVALUES
-2.0000 2.0000 -0.0000 -0.0000

EIGENVECTORS
-0.5000 0.5000 0.0 0.7071
0.5000 0.5000 0.7071 0.0
-0.5000 0.5000 0.0 -0.7071
0.5000 0.5000 -0.7071 0.0

Figure 4.2.1 A 4-point Graph
Thus the partial permutation

\[ R^* = (1, 2) \rightarrow (1, 2), R^*_i |(i, \ldots, n) = P^*_i |(i, \ldots, n) \]

is a proper partial permutation containing \( P^* \), contradicting the maximality of \( P^* \).

**Corollary 4.2.3A.** If \( G \) is isomorphic to \( H \) and \( P_1^*, \ldots, P_n^* \) are all the maximal proper partial permutations from \( G \) to \( H \), then

\[ \Gamma(G) = < < P_i^* >, P_i^* P_i^T >, i \text{ fixed, } 1 \leq j \leq n. \]

**Proof.** The number of automorphisms of \( G \) is equal to the number of isomorphisms from \( G \) to an isomorphic graph \( H \). All the maximal proper partial permutations from \( G \) to \( H \) induce the same partition on the nodes of \( G \), therefore each gives rise to the same number of isomorphisms, namely \( |< P_i^* >| \). Thus the total number of isomorphisms is \( k|< P_i^* >| \).

The number of automorphisms in \( < < P_i^* >, P_i^* P_i^T >, i \text{ fixed, } j=1, \ldots, k \) is also \( k|< P_i^* >| \). Since \( P_i^* P_i^T \neq P_j^* P_j^T \) if \( j \neq k \), all the elements of \( < < P_i^* >, P_i^* P_i^T > \) are distinct, and therefore this is all the group of \( G \).

**Corollary 4.2.3B.** If there exists a unique maximal proper partial permutation \( P^* \) from a graph \( G \) to an isomorphic graph \( H \), then \( \Gamma(G) = < P^* > \), and the automorphism partitioning of \( G \) is the domain of \( P^* \) (similarly for \( H \)).

**Proof.** This follows directly from theorem 4.2.2, since \( PP^T = I \).
Corollary 4.2.3C. Let G, H be isomorphic graphs. Let $P_1^*, \ldots, P_k^*$ be all the maximal proper partial permutations from G to H.

Let $P_i^*: s_i \rightarrow t_i, \ldots, s_{i_{k_i}} \rightarrow t_{i_{k_i}}$.

Let $U_{i_{k_i}} = \bigcup_{k_i} P_i^* P_{i_{k_i}}^*(S_{i_{k_i}})$.

Then the automorphism partitioning of G is $\{U_{i_1}, U_{i_2}, \ldots, U_{i_{k_i}}\}$.

Proof. Clearly, the elements of $U_{i_m}, m=1, \ldots, l$, belong to the same set of the automorphism partitioning, and the sets in $\{U_{i_1}, \ldots, U_{i_{k_i}}\}$ have a non-empty intersection, or are equal, by Theorem 4.2.3.

Theorem 4.2.4. Let G, H be isomorphic graphs with all distinct eigenvalues. Then the sets in the domain of any maximal proper partial permutation contain at most two elements.

Proof. By Theorem 4.2.2, if $P^*$ is a maximal proper partial permutation and $P^*$ sends $s=(a_1, a_2, \ldots, a_k)$ into $t=(b_1, b_2, \ldots, b_k)$, then $S^*_2(G)$. Mowshowitz [22] has shown that if a graph has all distinct eigenvalues, every element of its group has order 2, and we therefore must have $k \leq 2$.

4.3 Formal Description of the Algorithm

We have seen in the preceding section that our method can
be used, depending on the graphs we are dealing with, to find one isomorphism between two graphs, or to find all isomorphisms between them. We present here two versions of the algorithm. The first, Algorithm I, yields all the isomorphisms between two cospectral graphs whose eigenvalues are all distinct, and is meant to be used mostly to obtain the group of a graph with all distinct eigenvalues, since when comparing two graphs for isomorphism one is usually not interested in finding all the isomorphisms.

Algorithm I is essentially a breadth-first search: the eigenvectors of $G$ and $H$ corresponding to eigenvalue $i$ ($i=1,\ldots,n$) are compared to yield 0, 1 or 2 maximal partial permutations which map one onto the other. The partial permutations obtained at each stage are intersected with all those obtained previously, and the resulting partial permutations are the next candidates for proper partial permutations. The partial permutations obtained after all eigenvectors have been compared are proper, that is, all the permutations they contain are isomorphisms from $G$ to $H$.

The second version, Algorithm II, yields one partial permutation between the two graphs. It is a depth-first search: the eigenvectors belonging to eigenvalue $i$ ($i=1,\ldots,k$) are compared, one partial permutation is found, and it is intersected with the partial permutation obtained previously. If the intersection is empty, the Algorithm tries the next possible choice for a partial permutation. If none is found, it backs up to the next choice for the preceding eigenvalue, etc.
Both versions make use of Algorithm (a) to find the intersection of two partial permutations. Algorithm II also uses Algorithms (b) and (c) to obtain the next combination and next permutation of indices to be used when setting up a system of linear equations as restricted by a partial permutation P*.

**Algorithm (a):** finding the intersection of two partial permutations.

Given two partial permutations, P* and Q*, the Algorithm finds a maximal partial permutation R* s.t. R* is contained in P* and in Q*, i.e. R* = P* ∩ Q*.

Let P* = p₁ → l₁, ..., pₚ → lₚ, Q* = q₁ → m₁, ..., qₘ → mₘ. The members of the elements of the partitions are ordered by ascending value, the elements of the partitions are ordered by ascending value of their first element.

The partial permutation returned by the Algorithm is in the form R* = r₁ → n₁, ..., rₙ → nₙ.

**Step 1.** Set R* = φ

**Step 2.** If P* = φ, Q* = φ, return R*.

Else set p = p, q = q.

Delete p from {p₁}, delete q from {q₁}.

**Step 3.** If cardinality(p ∩ q) = cardinality(P*(p) ∩ Q*(q)) then set r = p ∩ q, set n = R*(r) = P*(p) ∩ Q*(q). Order r in {r₁}. Else, return R* = φ.
IV: THE EIGENVECTOR CONNECTION

Step 4. Set \( p = p \setminus (p \cap q) \), \( P^*(p) = P^*(p) \setminus (P^*(p) \cap Q^*(q)) \), order \( p \) in \( \{p\} \). Set \( q = p \setminus (g \cap p) \), \( Q^*(q) = Q^*(q) \setminus (P^*(p) \cap Q^*(q)) \), order \( g \) in \( \{g\} \). Go to Step 2.

**Algorithm (a)**: obtain the next permutation of \( r \) elements partitioned into \( k \) sets. The Algorithm is in the form of a recursive procedure, and we assume an algorithm for generating permutations on a set, next-permutation, (such as IPERM, written at the UBC Computing Center) is given.

Procedure next-perm-set \((k)\)

If \( k = 1 \) then next-permutation of set 1
else

If all permutations on \( k-1 \) sets have been generated, reset sets 1, 2, ..., \( k-1 \) to their original state, next-permutation of set \( k \)
else

next-perm-set \((k-1)\).

**Algorithm (c)**: obtain the next combination of \( k \) out of \( n \) elements partitioned into \( r \) sets, where a fixed number of elements, \( f(j) \), is taken from set \( j \), for all \( j \). An algorithm for generating combinations of \( i \) elements of a set, next-combination, (for example, algorithm 382 of the CACM) is assumed given.

Procedure next-comb-set \((k)\)

If \( k = 1 \) then next combination of \( f(1) \) elements of set 1
else
if all combinations on \( k-1 \) sets have been generated, reset the combinations of \( f(j) \) elements of set \( j, j=1, \ldots, k-1 \), to their original state, next-combination of \( f(k) \) elements of set \( k \).

Else

next-comb-set \((k-1)\).

Let \( G \) and \( H \) be cospectral connected graphs on \( n \) points, with distinct eigenvalues.
Let \( \lambda_1, \ldots, \lambda_n \) be their eigenvalues, ordered by increasing value.
Let \( x_1, \ldots, x_n \) be the eigenvectors of \( G \) corresponding to \( \lambda_1, \ldots, \lambda_n \),
\( y_1, \ldots, y_n \) be the eigenvectors of \( H \) corresponding to \( \lambda_1, \ldots, \lambda_n \).

**Algorithm I:** finding all isomorphisms between two cospectral graphs with all distinct eigenvalues.

The Algorithm compares \( x_i \) with \( y_i \), \( i=1, \ldots, n \), starting with the eigenvectors corresponding to the smallest eigenvalue, to find a positive and/or negative maximal partial permutation that maps \( x_i \) onto \( y_i \). The partial permutation(s) obtained is (are) intersected with those obtained by comparing \( x_j \) to \( y_j, j \neq i \). The Algorithm yields all the maximal proper partial permutations from \( G \) to \( H \). Step 3 deals with a special case, namely when \( x_i \) and \( y_i \) have only two non-zero components, and these have opposite signs and equal absolute value. In this case we can obtain both a positive and a negative partial permutation that map \( x_i \) onto \( y_i \), but neither would be maximal. We therefore introduce a special test (cf. Ex 4.1.3), so that the partial
permutation we get is indeed maximal.

Step 1. Set $i=1$. Set $m=1$. Set $P = (1,\ldots,n) \rightarrow (1,\ldots,n)$

Step 2. Sort the components of $x_i$ in $\text{sorted}(x_i)$, the components of $y_i$ in $\text{sorted}(y_i)$.

Step 3. If $x_i^{(t)} = a$, $x_i^{(w)} = -a$, $x_i^{(v)} = 0$, $t \neq j, k$ and $y_i^{(I)} = a$, $y_i^{(L)} = -a$, $y = 0$, $t \neq l, m$, then set $Q^* = (j,k) \rightarrow (l,m)$, $(a,\ldots,a) \rightarrow (b,\ldots,b)$, $a \neq j, k$, $b \neq l, m$. Set $R^* = \emptyset$, go to Step 7.

Step 4. If $\text{sorted}(x_i) = \text{sorted}(y_i)$, find a maximal partial permutation $Q^*$ such that $Qx_i = y_i$, for all permutations $Q$ refining $Q^*$ (by matching maximal sets of components of equal value). Else, set $Q^* = \emptyset$.

Step 5. If $\text{sorted}(x_i) = \text{reverse}(\text{sorted}(y_i))$, find a maximal partial permutation $R^*$ such that $Rx_i = -y_i$, for all permutations $R$ refining $R^*$. Else, set $R^* = \emptyset$.

Step 6. If $Q^* = \emptyset$ and $R^* = \emptyset$, stop, no isomorphism is possible.
IV: THE EIGENVECTOR CONNECTION

Step 7. Set $j = 1$

Step 8. If $P(j) \cap Q \neq \emptyset$, set $P(j)^+ = P(j) \cap Q$  
if $P(j) \cap R \neq \emptyset$, set $P(j)^- = P(j) \cap R$.

Step 9. Set $j = j + 1$. If $j < m$, go to Step 8.

Step 10. Arrange the distinct non-empty elements of $P(j)^+$,  
$P(j)^-$, $j=1,\ldots,m$ into a new list of possible partial  
permutation, $P(1)^*,\ldots,P(s)^*$.

Step 11. Set $m = s$.

If $m = 0$, stop, no isomorphism is possible.

Step 12. Set $i = i + 1$. If $i<n$, go to Step 2.

Else, the algorithm ends, yielding all the maximal  
proper partial permutations from $G$ to $H$,  
$P(1)^*,\ldots,P(m)^*$.

Algorithm II: comparing two cospectral graphs for isomorphism.

Let $G$ and $H$ be cospectral connected graphs on $n$ points.
Let $\lambda_1,\ldots,\lambda_k$ be their distinct eigenvalues, ordered by increasing  
multiplicity, and by increasing value for distinct eigenvalues  
of equal multiplicity.
Let $x_1,\ldots,x_k$ be the eigenvectors of $G$ corresponding to $\lambda_1,\ldots,\lambda_k$. 
y_1, ..., y_k be the eigenvectors of H corresponding to \lambda_1, ..., \lambda_k. Denote the multiplicity of \lambda_i, i=1, ..., t by m(i).
The Algorithm tries to match an eigenvector x of G corresponding to \lambda_i with a linear combination of the eigenvectors of H corresponding to \lambda_i. To solve the linear system involved, a set of rows of [y_1, ..., y_k] is chosen, which gives the coefficients of the system, and a set of components of x_i is chosen, which gives the right hand side of the system.
The choice is restricted by the current partial permutation i denotes the current eigenvalue, PAR the current partial permutation. The sets in PAR are ordered as in Algorithm (a).
SLE denotes the system of linear equations set up with elements of the eigenvectors of H corresponding to i, RHS its right-hand side, set up with elements of the first eigenvector of G corresponding to i.

Step 1 set i=1
set PAR(i)=(1, ..., n) -> (1, ..., n)
Step 2 if i>t, go to Step 15
Step 3 get next combination of m(i) indices for the linear system, as restricted by PAR(i) (i.e., choose the first m(i) indices of y from the union of the smallest sets of PAR that contains m(i) elements, then m(i)+1 elements, m(i)+2, etc.). If no choice is left, go to Step 12, else choose indices for elements of x that correspond to the indices chosen for y, as indicated by PAR.
Step 4 set up SLE with the chosen elements of \([y_1, \ldots, y_n]\).
Step 5 set up RHS with the chosen elements of \(x_i\).
Step 6 if RHS is the zero vector, go to Step 14
Step 7 solve the linear system. If no solution exists, go to Step 3.
Step 8 compare \(\text{order}(x_i)\) and the linear combination of \(y_i\)'s obtained to find a partial permutation MPP that maps one onto the other. Set \(\text{flag}(i)=1\). Go to Step 10.
Step 9 compare \(\text{order}(x)\) and the linear combination of \(y_i\)'s to find a negative partial permutation MPP that maps one onto the other. Set \(\text{flag}(i)=2\).
Step 10 if \(\text{MPP}\neq\emptyset\), get \(\text{TEMP} = \text{PAR}(i) \cap \text{MPP}\) (using Algorithm (a))
      if \(\text{MPP}\neq\emptyset\) or \(\text{TEMP}\neq\emptyset\) go to Step 13
Step 11 \(i=i+1\)
Step 12 \(\text{PAR}(i) = \text{TEMP}\)
      go to Step 2
Step 13 \(i=i-1\)
      if \(i=0\), fail
Step 14 if \(\text{flag}(i)=1\) go to Step 9.
      If \(\text{flag}(i)=2\) set \(\text{flag}(i)=0\).
Step 15 get next permutation of RHS (using Algorithm (b)). If none exists, go to Step 16. Else go to Step 7.
Step 16 get next combination of indices for RHS (using Algorithm (c)). If none exists, go to Step 11. Else go to Step 6.
Step 17 check whether \(\text{TEMP}\) contains an isomorphism
      get next permutation contained in \(\text{TEMP}\) (using Algorithm
(b)).

If none exists, go to Step 15.
Else check whether the current permutation is an isomorphism. If it is, stop, else, go to Step 17

4.4 Order of the computation

It is very difficult to give a good estimate of the order of the computation for either Algorithm I or Algorithm II, as is the case in general for heuristic algorithms. The obvious bounds are in general far too pessimistic.

Let G be a graph on n nodes whose eigenvalues are all distinct. If we use algorithm I to find its group, we may in principle obtain two maximal partial permutations for each eigenvalue, and if each of these yielded a new partial permutation when intersected with those obtained previously, we would obtain, for eigenvalue \( k \), \( 2^k \) distinct maximal partial permutations mapping the first \( k \) eigenvectors of G to themselves. In particular, we would end up with \( 2^n \) proper maximal partial permutations, yielding at least \( 2^n \) automorphisms. Such a computation would obviously not be efficient. Mowshowitz [24] has shown that the order of the group of a graph whose eigenvalues are all distinct is at most \( 2^{\binom{n}{2}} \). Thus it is clearly impossible to obtain \( 2^n \) distinct maximal proper partial permutations.
**IV: THE EIGENVECTOR CONNECTION**

**EIGENVALUES**

-2.1774 -1.0000 0.0000 0.3216 2.8558

**EIGENVECTORS**

\[
\begin{align*}
0.5244 & \quad -0.0000 & 0.7071 & \quad 0.1312 & \quad -0.4558 \\
0.5244 & \quad 0.0000 & -0.7071 & \quad 0.1312 & \quad -0.4558 \\
-0.3301 & \quad -0.7071 & \quad -0.0000 & \quad -0.3870 & \quad -0.4912 \\
-0.3301 & \quad 0.7071 & \quad 0.0000 & \quad -0.3870 & \quad -0.4912 \\
-0.4817 & \quad 0.0000 & \quad 0.0000 & \quad 0.8161 & \quad -0.3192
\end{align*}
\]

\[P^*: (1,2) \rightarrow (1,2), (3,4) \rightarrow (3,4), 5 \rightarrow 5\]

\[\Gamma(G) = \langle (1,2), (3,4), (5) \rangle\]

*Figure 4.5.7* A 5-point Graph with Simple Eigenvalues
in fact, since our algorithm yields generators for \( \Gamma(G) \), rather than all the elements of \( \Gamma(G) \) (i.e., partial permutations rather than permutations), even \( 2^{2^{c_1}} \) is in general too high a bound. For example, the graph of Fig 4.5.1 (used as an example by Mowshowitz) has 4 automorphisms, but our algorithm yields only one partial permutation which generates the group. Thus, in practice, the number of maximal proper partial permutations obtained is often far less than the order of the group.

Mapping one eigenvector onto the other involves computation of order \( n \log n \) (since it consists of ordering the vectors and comparing them, and ordering the resulting partial permutation). Similarly, intersecting two partial permutations can be done in a number of steps of order \( n^2 \). Thus as long as the number of partial permutations is small, the computation will be efficient. In all the examples we tried, the number of partial permutations generated was far smaller than \( 2^{\binom{n}{2}} \), and even though partial permutations may be generated at some stage which are later rejected, the computation was fast in all cases.

We next look at the computation involved in Algorithm II. Let \( \lambda \) be an eigenvalue of multiplicity \( k \). We first have to find a system of \( k \) linear equations which is linearly independent (Step 3). In the worst case, to set up the linear system, we will be choosing \( k \) out of \( n \) elements, and therefore might have to try \( \binom{n}{k} \) systems. Solving the system is of the order of \( \lambda^3 \), so that altogether we might need \( o(\binom{n}{k} \lambda^3) \) steps to find a system which has a solution. Usually, however, the choices for the linear system will be far more restricted, as
soon as a non-trivial partial permutation is obtained. In practice, a linearly independent system is often found immediately.

In order to find a mapping of the eigenvectors, we then compare $\text{sort}(x)$ to $\text{sort}\left(\sum_{x} a_{i} y_{i}\right)$, where $x$, $y_{i}$ belong to $\lambda$. If this does not yield a mapping, or if the partial permutation obtained does not agree with the one obtained previously, we try to solve the same system again with a different ordering of its right hand side. If the system uses $l_{i}$ elements of $i$ sets of the current partial permutation, the number of possible permutations is $\prod_{x} l_{x}!$. For each such choice, we solve the system again. Only back substitution is necessary, therefore if we have to try all permutations, the computation is of order $\prod_{x} l_{x}! k$. If no permutation of the right-hand-side of the linear system yields a solution, we next try to use a different combination of elements of $x$. Assume the system uses $l_{i}$ elements of $i$ sets of the current partial permutation, where set $j$ has $c_{j}$ elements. Then there are $\binom{c_{j}}{l_{i}}$ possible ways of choosing the $l_{i}$ elements, for each $j$, and therefore we have $\prod_{x} \binom{c_{j}}{l_{i}}$ possible combinations of elements of $x$ for the right hand side of our system of equations. For each of these combinations, we have to try all the possible permutations of the chosen elements. We therefore may have to try and solve the system with $\prod_{x} \binom{c_{j}}{l_{i}} \prod_{j} l_{j}!$ different right hand sides, at worst $\binom{c_{j}}{l_{i}} k!$. This is for example the case with the cospectral graphs of Appendix II. The only eigenvalue of multiplicity one does not yield a non-trivial partial permutation, since the graph is
regular. The next eigenvalue has multiplicity three. Since the graphs are not isomorphic, we repeatedly back up to the second root, in an attempt to find a mapping between the eigenvectors that can also be used with the following roots. For the second root, there are \(\binom{1}{3} \binom{2}{5}\) possible choices for a right hand side, once a linearly independent system has been found. Each of these has to be tried before returning the answer that the graphs are not isomorphic, after about 20 seconds. This example, however, is a case of a worst possible graph for our algorithm. In all other examples we tried, if the graphs were cospectral but not isomorphic, this was discovered very rapidly, with virtually no backtracking. Similarly, for isomorphic graphs, an answer (that is, a permutation specifying the isomorphism) was usually returned with little backtracking.

For graphs whose eigenvalues have low multiplicities, and particularly for graphs whose eigenvalues are all distinct the algorithm was found to be very efficient, that is, very little backing up was required. The algorithm is also very efficient if the group of the graph is large. For example, the complete graph on \(n\) points has two distinct eigenvalues, of multiplicity 1 and \(n-1\) respectively. Comparing the eigenvectors belonging to the first eigenvalue will not refine the initial partial permutation \((1, \ldots, n) \rightarrow (1, \ldots, n)\), since the graph is regular. However, comparing the eigenvectors belonging to the second eigenvalue will yield a solution immediately. Thus the efficiency of the algorithm seems directly related to the group of the graphs. Note that algorithms that try to distinguish
between nodes by looking at their attributes would have difficulty in matching two complete graphs.

The efficiency of our algorithm then seems to depend mainly on two factors: the multiplicity of the eigenvalues, and the richness of the group of the graph. The relationship between these two factors is poorly understood by the author, and therefore nothing more will be said on the subject.

4.5 Conclusion

We have discussed in this thesis the use of the eigenvalues and eigenvectors of the adjacency matrices of cospectral graphs in finding graph isomorphism. We presented two basic algorithms, essentially a breadth first and a depth first search, which use this method. In both cases, various devices can be used to speed up the computation. For example, if at any point a permutation is obtained, it is useless to keep trying to refine it, instead one should check directly for isomorphism. Similarly if a partial permutation is obtained which contains only a small number of permutations, etc. We intend to look in more detail at various strategies for minimizing backtracking in our algorithms. For example, it might be worthwhile (this can only be ascertained by extensive testing) to first check whether there exists a mapping of the first eigenvector of G belonging to \( \lambda_i \) to a linear combination of the eigenvectors of H belonging to \( \lambda_i \), for each i, before trying to find one mapping which works for all i (cf. Mackworth [20]).
Algorithm II is very efficient when dealing with graphs with simple eigenvalues. The reason is that in this case there can be only 2 mappings of the vectors (if they are normalized), while if the eigenvalue has multiplicity $k$, we may have to try up to $\binom{n}{k}k!$ mappings. One way to retain the efficiency of the algorithm in the case where the eigenvalues are not simple would be to derive the eigenvectors in such a way that if

$$P^\top AP = B$$

then $x_i = Py_i$, for $i=1,\ldots,n$,

i.e., use an algorithm for computing the eigenvectors of a matrix which is invariant under permutations. It is not clear whether this is possible, but if such an algorithm could be found, and be as efficient and as accurate as those currently used, we would be well on our way to finding an efficient general purpose graph isomorphism algorithm.


12. FISHER, M. E. On hearing the shape of a drum. J. of


15. HARARY, F. Graph Theory, Addison-Wesley, Reading, 1969.


22. MOWSHOWITZ, A. The group of a graph whose adjacency matrix


31. UNGER, S. H. GIT - a heuristic program for testing pairs of directed line graphs for isomorphism. C.A.C.M., 7 (1964)
BIBLIOGRAPHY

pp. 26-34.


APPENDIX A Examples of Cospectral Graphs

EIGENVALUES

\[-2.0840 \quad -1.5718 \quad -1.0000 \quad -0.4317 \quad 0.0 \quad 0.4317 \quad 1.0000 \quad 1.5718\]

EIGENVALUES

2.0840

EIGENVECTORS

\[
\begin{array}{cccccccc}
-0.1919 & -0.3626 & 0.0 & -0.2858 & 0.7071 & 0.2858 & 0.0 & -0.3626 \\
-0.1919 & -0.3626 & 0.0 & -0.2858 & 0.7071 & 0.2858 & 0.0 & -0.3626 \\
0.3999 & 0.5699 & 0.0 & 0.1234 & 0.0 & 0.1234 & 0.0 & 0.5699 \\
-0.4496 & -0.1707 & 0.0 & 0.5184 & 0.0 & -0.5184 & 0.0 & -0.1707 \\
0.5371 & -0.3017 & 0.0 & -0.3472 & 0.0 & -0.3472 & 0.0 & 0.3017 \\
-0.3348 & 0.3224 & -0.5000 & -0.1842 & 0.0 & 0.1842 & 0.5000 & 0.3224 \\
0.1607 & -0.2051 & 0.5000 & 0.4267 & 0.0 & 0.4267 & 0.5000 & 0.2051 \\
-0.3348 & 0.3224 & 0.5000 & -0.1842 & 0.0 & 0.1842 & -0.5000 & 0.3224 \\
0.1607 & -0.2051 & -0.5000 & 0.4267 & 0.0 & 0.4267 & -0.5000 & 0.2051 \\
\end{array}
\]

EIGENVECTORS

0.1919

0.1919

0.3999

0.4496

0.5371

0.3348

0.1607

0.3348

0.1607

(1) Smallest Cospectral Trees with Identical Degree Partitioning ([16])
APPENDIX A Examples of Cospectral Graphs

EIGENVALUES

-2.0840 -1.5718 -1.0000 -0.4317 0.0 0.4317 1.0000 1.5718

EIGENVALUES

2.0840

EIGENVECTORS

\[-0.2367 -0.2280 -0.3536 -0.1303 0.7071 -0.1303 -0.3536 0.2280\]
\[-0.2367 -0.2280 -0.3536 -0.1303 -0.7071 -0.1303 -0.3536 -0.2280\]
\[0.4934 0.3584 0.3536 0.0562 -0.0000 -0.0562 -0.3536 0.3584\]
\[-0.5547 -0.1073 0.3536 0.2363 0.0000 0.2363 0.3536 0.1073\]
\[0.2662 0.0683 -0.3536 -0.5473 0.0000 0.5473 0.3536 0.0683\]
\[0.3964 -0.2580 -0.3536 0.3890 -0.0000 -0.3890 0.3536 -0.2580\]
\[-0.2714 0.5128 -0.0000 -0.4042 -0.0000 -0.4042 -0.0000 -0.5128\]
\[0.1692 -0.5481 0.3536 -0.2145 0.0000 0.2145 -0.3536 -0.5481\]
\[-0.0812 0.3487 -0.3536 0.4968 0.0000 0.4968 -0.3536 -0.3487\]

EIGENVECTORS

0.2367
0.2367
0.4934
0.5547
0.2662
0.3964
0.2714
0.1692
0.0812

(1) Smallest Cospectral Trees with Identical Degree Partitioning ([16])
APPENDIX A Examples of Cospectral Graphs

EIGENVALUES

-2.0000 2.0000 0.0 0.0 0.0

EIGENVECTORS

<table>
<thead>
<tr>
<th>0.0  0.0  0.0  0.0  1.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5000 0.5000 0.0 0.7071 0.0</td>
</tr>
<tr>
<td>0.5000 0.5000 0.7071 0.0 0.0</td>
</tr>
<tr>
<td>-0.5000 0.5000 -0.7071 0.0 0.0</td>
</tr>
<tr>
<td>0.5000 0.5000 -0.7071 0.0 0.0</td>
</tr>
</tbody>
</table>

(2) Smallest Cospectral Graphs ([16])
APPENDIX A Examples of Cospectral Graphs

EIGENVALUES

-1.9032 0.1939 1.0000 2.7093 -1.0000 -1.0000

EIGENVECTORS

0.2603 0.2136 -0.5000 -0.3696 0.3386 0.6208
0.2603 0.2136 -0.5000 -0.3696 0.3386 -0.6208
-0.7558 -0.1721 0.0 0 -0.6318 0.0 0.0
0.2603 0.2136 0.5000 -0.3696 -0.6208 0.3386
0.2603 0.2136 0.5000 -0.3696 0.6208 -0.3386
0.3971 -0.8877 0.0 -0.2332 0.0 0.0

(3) Smallest Cospectral Connected Graphs ([16])
APPENDIX A Examples of Cospectral Graphs

EIGENVALUES

\[-2.2725 \quad -1.7320 \quad -1.4924 \quad -0.7801 \quad 0.7801 \quad 1.4924 \quad 1.7320 \quad 2.2725\]

EIGENVALUES

\[-0.0000 \quad -0.0000 \quad -0.0000 \quad -0.0000\]

EIGENVECTORS

\[
\begin{bmatrix}
0.1180 & 0.2887 & -0.2738 & -0.1350 & -0.1350 & 0.2738 & 0.2887 & -0.1180 \\
0.1180 & 0.2887 & -0.2738 & -0.1350 & -0.1350 & 0.2738 & 0.2887 & -0.1180 \\
-0.2681 & -0.5000 & 0.4087 & 0.1053 & -0.1053 & 0.4087 & 0.5000 & -0.2681 \\
0.3734 & 0.2887 & -0.0622 & 0.1879 & 0.1879 & 0.0622 & 0.2887 & -0.3734 \\
-0.5803 & 0.0 & -0.3158 & -0.2519 & 0.2519 & -0.3158 & 0.0 & -0.5803 \\
0.2554 & 0.0 & 0.2116 & 0.3229 & 0.3229 & -0.2116 & 0.0 & -0.2554 \\
0.3167 & 0.0 & 0.3841 & -0.5022 & -0.5022 & -0.3841 & 0.0 & -0.3167 \\
-0.1394 & 0.0 & -0.2574 & 0.6437 & -0.6437 & -0.2574 & 0.0 & -0.1394 \\
0.3734 & -0.2887 & -0.0622 & 0.1879 & 0.1879 & 0.0622 & -0.2887 & -0.3734 \\
-0.2681 & 0.5000 & 0.4087 & 0.1053 & -0.1053 & 0.4087 & 0.5000 & -0.2681 \\
0.1180 & -0.2887 & -0.2738 & -0.1350 & -0.1350 & 0.2738 & -0.2887 & -0.1180 \\
0.1180 & -0.2887 & -0.2738 & -0.1350 & -0.1350 & 0.2738 & -0.2887 & -0.1180
\end{bmatrix}
\]

EIGENVECTORS

\[
\begin{bmatrix}
-0.0996 & 0.7071 & 0.1261 & -0.3053 \\
-0.0996 & -0.7071 & 0.1261 & -0.3053 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.1992 & 0.0 & -0.2522 & 0.6107 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.2865 & 0.0 & -0.2357 & -0.6586 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
-0.4857 & 0.0 & 0.4878 & 0.0479 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.7451 & 0.0 & 0.2521 & 0.0170 \\
-0.2594 & 0.0 & -0.7400 & -0.0650
\end{bmatrix}
\]

(4) Non-isomorphic Generalized Cospectral Trees ([30])
APPENDIX A Examples of Cospectral Graphs

EIGENVALUES
-2.2725 -1.7320 -1.4924 -0.7801 0.7801 1.4924 1.7320 2.2725

EIGENVALUES
-0.0000 -0.0000 -0.0000 -0.0000

EIGENVECTORS
0.2337 0.2041 0.1758 -0.1986 -0.1986 0.1758 -0.2041 0.2337
0.2337 0.2041 0.1758 -0.1986 -0.1986 0.1758 -0.2041 0.2337
0.2337 0.2041 0.1758 -0.1986 -0.1986 0.1758 -0.2041 0.2337
-0.5312 -0.3536 -0.2624 0.1550 -0.1550 0.2624 -0.3536 0.5312
0.5059 -0.0000 -0.1359 0.4750 0.4750 -0.1359 0.0000 0.5059
-0.3255 0.3536 -0.0268 -0.3543 0.3543 0.0268 0.3536 0.3255
0.2337 -0.6124 0.1758 -0.1986 -0.1986 0.1758 0.6124 0.2337
-0.1029 0.3536 -0.1178 0.2546 -0.2546 0.1178 0.3536 0.1029
-0.1029 0.3536 -0.1178 0.2546 -0.2546 0.1178 0.3536 0.1029
-0.2930 0.0000 0.4919 -0.1713 0.1713 -0.4919 0.0000 0.2930
0.1599 -0.0000 -0.5982 -0.3414 -0.3414 -0.5982 0.0000 0.1599
-0.0704 0.0000 0.4009 0.4376 -0.4376 -0.4009 0.0000 0.0704

EIGENVECTORS
-0.0000 0.4082 0.7054 -0.0485
-0.0000 0.4082 -0.7054 0.0485
0.0000 -0.8165 0.0000 -0.0000
0.0000 -0.0000 0.0000 0.0000
0.0000 -0.0000 0.0000 -0.0000
0.0000 -0.0000 0.0367 0.5333
-0.0000 0.0000 0.0000 0.0000
-0.7071 0.0000 -0.0183 -0.2666
-0.7071 0.0000 -0.0183 -0.2666
-0.0000 0.0000 -0.0367 -0.5333
-0.0000 0.0000 0.0000 -0.0000
0.0000 -0.0000 0.0367 0.5333

(4) Non-isomorphic Generalized Cospectral Trees ([30])
APPENDIX A Examples of Cospectral Graphs

EIGENVALUES

-2.7152 -1.2758 -1.0000 1.0000 1.2758 2.7152 0.0000 0.0000

EIGENVALUES

-0.0000

EIGENVECTORS

\[
\begin{array}{cccccccc}
-0.1674 & -0.4246 & 0.0 & 0.0 & 0.4246 & 0.1674 & 0.7071 & -0.0474 \\
-0.1674 & -0.4246 & 0.0 & 0.0 & 0.4246 & 0.1674 & -0.7071 & -0.0474 \\
0.4544 & 0.5418 & 0.0 & 0.0 & 0.5418 & 0.4544 & 0.0 & 0.0 \\
-0.4495 & 0.0790 & 0.0 & 0.0 & -0.0790 & 0.4495 & 0.0 & 0.6501 \\
0.3831 & -0.3213 & 0.5000 & 0.5000 & -0.3213 & 0.3831 & 0.0 & 0.0 \\
-0.1411 & 0.2518 & -0.5000 & 0.5000 & -0.2518 & 0.1411 & 0.0 & -0.0949 \\
-0.4495 & 0.0790 & 0.0 & 0.0 & -0.0790 & 0.4495 & 0.0 & 0.7449 \\
0.3831 & -0.3213 & -0.5000 & -0.5000 & -0.3213 & 0.3831 & 0.0 & 0.0 \\
-0.1411 & 0.2518 & 0.5000 & -0.5000 & -0.2518 & 0.1411 & 0.0 & -0.0949 \\
\end{array}
\]

EIGENVECTORS

\[
\begin{array}{cccccccc}
0.2848 \\
0.2848 \\
0.0 \\
-0.4010 \\
0.0 \\
0.5695 \\
-0.1685 \\
0.0 \\
0.5695 \\
\end{array}
\]

(5) Non-Isomorphic Generalized Cospectral Connected Graphs ([30])
APPENDIX A Examples of Cospectral Graphs

EIGENVALUES

-2.7152 -1.2758 -1.0000 1.0000 1.2758 2.7152 -0.0000 -0.0000

EIGENVALUES

-0.0000

EIGENVECTORS

-0.1781 -0.0998 -0.5000 0.5000 0.1781 0.1995 0.4835 -0.1291 0.2610
0.3562 0.1273 0.5000 -0.5000 0.3562 0.1273 0.4835 -0.2582 -0.7746
0.1273 -0.1995 0.5000 -0.5000 -0.1273 -0.1995 -0.4835 -0.0000 -0.0000
-0.1781 -0.0998 0.5000 -0.5000 0.5000 0.1781 0.4835 0.1273 -0.2610
-0.1291 0.2610 0.5000 0.5000 0.1273 -0.4835 -0.0000 -0.0000 -0.0000
-0.4225 -0.4225 -0.0000 0.0000 0.0000 -0.4225 -0.4225 0.6455 -0.0000
0.1800 0.1800 0.0000 0.0000 0.0000 -0.1800 0.0000 0.0000 -0.0000
-0.0663 0.0663 0.0000 0.0000 0.0000 -0.0663 -0.6455 0.0000 -0.0000

EIGENVECTORS

0.5761
-0.0000
0.0038
0.0000
0.5761
-0.5799
0.0000
-0.0000
-0.0000

(5) Non-Isomorphic Generalized Cospectral Connected Graphs ([30])
APPENDIX B A Nasty Pair of Cospectral Graphs

EIGENVALUES

\begin{align*}
4.0000 & -0.0000 -0.0000 -0.0000 & 2.0000 & 2.0000 & 2.0000 & -2.0000 \\
-2.0000 & -2.0000 & -2.0000 & -2.0000
\end{align*}

EIGENVALUES

\begin{align*}
-2.0000 & -2.0000 & -2.0000 & -2.0000
\end{align*}

EIGENVECTORS

\begin{align*}
0.2887 & -0.0184 & -0.0071 & -0.4996 & 0.0 & 0.5000 & 0.0 & 0.1899 \\
-0.2887 & 0.4907 & -0.0943 & -0.0167 & 0.3402 & 0.2500 & -0.2679 & -0.0461 \\
-0.2887 & 0.0940 & 0.4910 & -0.0104 & 0.3660 & 0.2500 & 0.2314 & -0.1438 \\
-0.2887 & -0.4907 & 0.0943 & 0.0167 & -0.3660 & 0.2500 & -0.2314 & -0.0664 \\
-0.2887 & 0.0184 & 0.0071 & 0.4996 & -0.0258 & 0.0 & -0.4993 & -0.2479 \\
-0.2887 & 0.0940 & 0.4910 & -0.0104 & -0.3660 & -0.2500 & -0.2314 & 0.3142 \\
-0.2887 & -0.0940 & -0.4910 & 0.0104 & -0.3402 & 0.2500 & 0.2679 & -0.1235 \\
-0.2887 & 0.0184 & 0.0071 & 0.4996 & 0.0258 & 0.0 & 0.4993 & -0.2018 \\
-0.2887 & 0.4907 & -0.0943 & -0.0167 & -0.3402 & -0.2500 & 0.2679 & 0.3253 \\
-0.2887 & -0.0940 & -0.4910 & 0.0104 & 0.3402 & -0.2500 & -0.2679 & 0.2940 \\
-0.2887 & -0.4907 & 0.0943 & 0.0167 & 0.3660 & -0.2500 & 0.2314 & 0.3455 \\
-0.2887 & -0.0184 & -0.0071 & -0.4996 & 0.0 & -0.5000 & 0.0 & -0.6395
\end{align*}

EIGENVECTORS

\begin{align*}
0.4448 & -0.1093 & 0.0370 & 0.4116 \\
0.1023 & -0.0531 & 0.1257 & -0.6208 \\
-0.5472 & 0.1623 & -0.1627 & 0.2092 \\
0.0003 & 0.5506 & -0.0653 & -0.3238 \\
0.0135 & -0.1021 & 0.3853 & 0.4429 \\
-0.0138 & -0.4485 & -0.3199 & -0.1192 \\
-0.4451 & -0.4413 & 0.0284 & -0.0878 \\
0.5167 & -0.0072 & -0.3284 & -0.0328 \\
-0.0716 & 0.4485 & 0.3000 & 0.1207 \\
-0.1159 & 0.1551 & -0.5110 & 0.1779 \\
0.0305 & -0.1551 & 0.4911 & -0.1764 \\
0.0854 & 0.0 & 0.0199 & -0.0015
\end{align*}

Non-isomorphic Cospectral Graphs
APPENDIX E A Nasty Pair of Cospectral Graphs

EIGENVALUES

4.0000 -0.0000 -0.0000 -0.0000 2.0000 2.0000 2.0000 -2.0000

EIGENVALUES

-2.0000 -2.0000 -2.0000 -2.0000

EIGENVECTORS

-0.2887 0.0000 -0.0000 -0.5000 -0.2167 -0.3749 0.2500 -0.0855
-0.2887 0.0000 0.0000 -0.5000 0.4257 -0.0793 0.2500 -0.0979
-0.2887 0.0000 -0.0000 0.5000 -0.2167 -0.3749 0.2500 -0.0872
-0.2887 -0.0000 0.5000 0.0000 -0.2167 -0.3749 -0.2500 0.3435
-0.2887 -0.0000 0.5000 0.0000 0.4257 -0.0793 -0.2500 0.1250
-0.2887 -0.0000 0.5000 0.0000 0.4257 -0.0793 -0.2500 0.3542
-0.2887 -0.5000 -0.0000 -0.0000 -0.2090 0.4542 0.0000 -0.3066
-0.2887 -0.5000 -0.0000 -0.0000 -0.2090 0.4542 0.0000 -0.4543
-0.2887 0.5000 0.0000 0.0000 0.0000 -0.0000 -0.5000 0.1233
-0.2887 0.5000 0.0000 0.0000 0.0000 -0.0000 -0.5000 0.6376

EIGENVECTORS

0.0087 0.6092 0.1929 0.0313
-0.3083 -0.5577 -0.0012 0.0307
-0.0699 -0.2127 0.1084 0.5892
-0.2297 0.2641 0.0834 -0.5272
0.2460 -0.2924 0.0475 -0.3878
0.1149 -0.1555 -0.5406 -0.2948
0.3533 0.1896 -0.4310 0.2638
0.4844 0.0527 0.1571 0.1707
0.1928 -0.1337 0.5088 -0.0936
-0.3614 0.0482 0.2711 0.0626
-0.4924 0.1851 -0.3170 0.1556
0.0617 0.0032 -0.0793 -0.0006

Non-isomorphic Cospectral Graphs