JOURNAL OF RESEARCH of the National Bureau of Standards—A. Mathematical Sciences
Vol. 80B, No. 4, October–December 1976

An Efficient Linear Algebraic Algorithm for the Determination of Isomorphism in Pairs of Undirected Graphs

Charles R. Johnson* and Frank Thomson Leighton

Institute for Basic Standards, National Bureau of Standards, Washington, D.C. 20234

(September 29, 1976)

An algorithm, complete with a specific FORTRAN implementation, is presented for the problem of determining whether or not two undirected graphs are isomorphic. The algorithm, centered upon the eigenvalues and eigenvectors of a modified adjacency matrix and techniques for decreasing the size of the automorphism group, is quite different from others (most of which are combinatorially based) and tends to work relatively very quickly on difficult test cases as well as on typical examples. Complexity estimates are given for many eventualities.

Key words: Graph isomorphism; labels; modified adjacency matrix; spectrum.

1. Introduction

Two undirected graphs are said to be *isomorphic* if there exists a one-to-one correspondence between their nodes which preserves adjacency. A computationally difficult issue in the theory of graphs is the determination of isomorphism. For example the graphs in figure 1a are pairwise isomorphic while those in 1b are not.

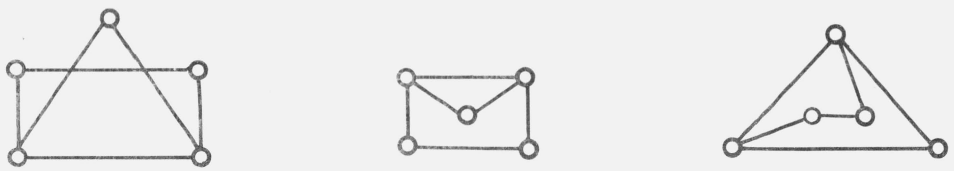

Figure 1a

Figure 1b

AMS Subject Classification: 68A10 05C99, 65F99.

* Present address: Institute for Physical Science and Technology, University of Maryland, College Park, Md. 20742.

The ability to divide a set of graphs into maximal subsets whose elements are pairwise isomorphic has several immediate applications. If nodes are identified with atoms of a particular compound and if edges are identified with the existence of a chemical bond, then, for instance, such an ability is of use to the chemist in distinguishing among many theoretically produced compounds. Other applications include physics (e.g. "hearing" the shape of a drum [1])[1], electronic circuit theory, linguistics etc.

Much effort has been devoted to the development of an algorithm which will determine whether or not any two graphs are isomorphic in an amount of time that is bounded by a finite polynomial in the number of nodes of the graphs considered. At the moment, the authors are unaware of the existence of any such algorithm. Many approximate or heuristic procedures have been developed however. Many such algorithms check conditions combinatorially necessary for isomorphism, some also attempt to construct a permutation that would exhibit isomorphism. Corneil presents a review of many such techniques. All known exact methods grow exponentially in required time. Unfortunately, as pointed out in [4] such procedures could take very long to determine whether or not two graphs of order 15 are isomorphic. Hence, such methods are clearly inadequate for large order graphs.

If a computer is to be used to determine whether or not two graphs are isomorphic, the graphs must be represented in a form suitable for machine processing. The adjacency matrix of a graph is such a standard form. First, the nodes of the graph are arbitrarily associated with the integers 1,2,3 . . . n where n is the order of the graph. The (i,j) component of the adjacency matrix is then defined to be 1 if there is an edge connecting the ith node and the jth node and is 0 otherwise. The (i,i) entry commonly contains the label given the node. In the case of an unlabeled graph, (i,i) entries are typically zero. While different orderings of the nodes produce different adjacency matrices, two adjacency matrices are isomorphic if and only if, independent of labeling, they correspond to the same graph. Figure 2 illustrates this.

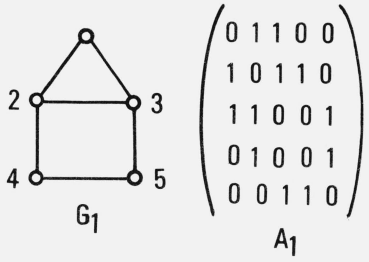

Figure 2a

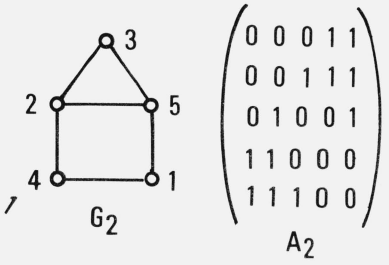

Figure 2b

$$A_2 = P^T A_1 P \text{ Where } P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 2c

The algorithm about to be described is divided roughly into three sections each satisfying a specific purpose. The first section consists of several basic tests for nonisomorphism. These tests are all $0(n^3)$ or better and are surprisingly effective in distinguishing nonisomorphic graphs. Since nearly all pairs of graphs that pass through the initial tests undistinguished are likely to be isomorphic, the second part of the algorithm involves the attempted construction of a permutation between the adjacency matrices of the graphs. In the great majority of cases, this is easily and quickly accomplished. The second section simultaneously serves as a more complicated test for nonisomorphism should it be found impossible to generate a satisfactory permutation.

[1] Figures in brackets indicate the literature references at the end of this paper.

The final section is specifically designed to deal with certain classes of matrices that do not easily yield to the tests or efforts to construct a permutation found in the previous two components. In general the tests in this section are more time consuming, generally $0(n^4)$, and are quite detailed. Methods of finding a satisfactory permutation are included as well as tests for nonisomorphism.

A complete FORTRAN listing of the algorithm, including explanatory comments, follows its description in section 3 and the test results provided in section 4 demonstrate the efficiency of the algorithm.

2. The Algorithm

Initially, the algorithm employs four basic tests, each of which checks for the failure of a condition necessary for isomorphism. The first test determines whether or not the node labels of the two graphs are identical. If they are not, then the corresponding graphs are not isomorphic. In the case of unlabeled graphs, the test yields no information and is unnecessary. This test is performed by SUBROUTINE LABEL and takes $0(n^2)$ amount of time when n is the order of the graphs.

A more useful test involves the comparison of the node valences of the two graphs. The valence of a node is the number of edges connected to it. The valence of each node, once computed, is combined with the node's label to form a new label for that node. Two nodes of a graph will have the same new label if and only if they have the same valence as well as identical initial labels. Thus two graphs with different sets of new labels are themselves nonisomorphic. For example, the graphs in figure 3 share the same set of node valences and of initial labels yet when these two qualities are considered mutually, the graphs are found to be nonisomorphic.

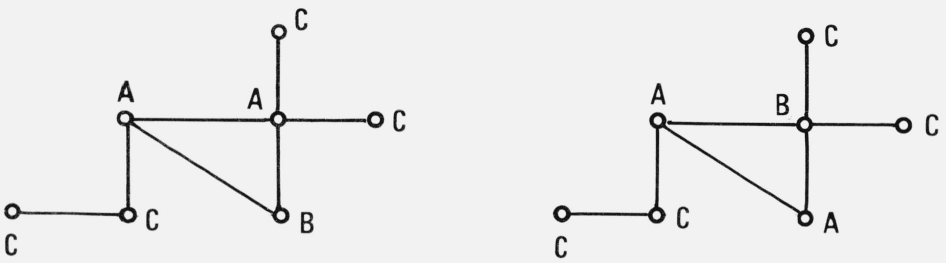

Figure 3

The set of initial node labels is (A,A,B,C,C,C,C) for each graph. Similarly the set of node valences is (1,1,1,2,2,3,4) in each case. When combined to produce sets of new labels, however, the following two distinct sets are formed: (A3,A4,B2,C1,C1,C1,C2) and (A2,A3,B4,C1,C1,C1,C2). Since these two sets are clearly not identical, the two graphs are not isomorphic. This test takes $0(n^2)$ time and is performed in SUBROUTINE VALENC.

The third basic test involves the search for and labeling of duplicate nodes. A given node is called an ordinary duplicate of degree k if there exist $k - 1$ and no more than $k - 1$ other identically labeled nodes in the graph which are connected by edges to the same nodes as is the given node. A given node is called a connected duplicate of degree k if the $k - 1$ nodes just mentioned are all mutually connected. To further illustrate consider the graph in figure 4. The numbers are only for use in referencing the nodes while the letters correspond to actual labels.

Nodes 1 and 3 are connected duplicates of degree 2. Nodes 2 and 4 are ordinary duplicates of degree 2. Nodes 6 and 7 are not duplicates since they have different labels. Finally nodes 10, 11, and 12 are ordinary duplicates of degree 3. All other nodes are trivially duplicates of degree 1.

Node duplicate degrees are easily found from the 0,1 adjacency matrix of a graph. Groups of ordinary duplicate nodes correspond to groups of identical rows (with identical labels) of the adjacency matrix with 0's entered on the diagonal. If 1's are entered on the diagonal, then the connected duplicates can be similarly found. It is easily seen that two graphs are not isomorphic if the ordinary and connected duplicate degrees of their nodes differ in any manner.

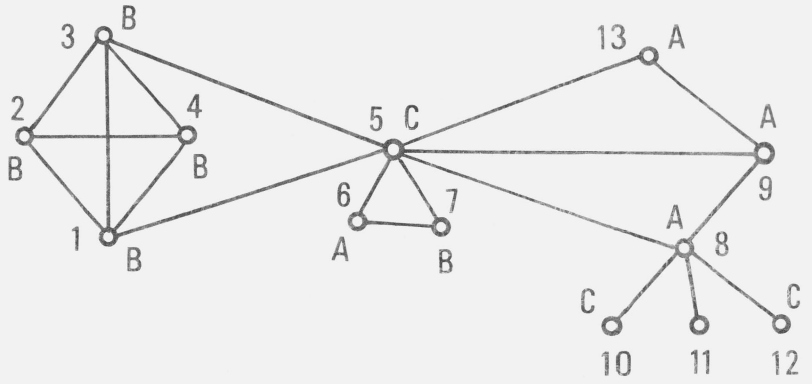

Figure 4

As before, the node labels of each graph are updated to reflect the newly obtained information and then compared. This is done in SUBROUTINE DUPLIC and takes $0(n^3)$ time.

While the identification of duplicate nodes serves as a worthwhile test for nonisomorphism, a potentially much more useful purpose is also served. A graph with a large number of nodes with duplicate degrees greater than one yields an adjacency matrix, A, and a large number of nontrivial automorphisms, that is permutations P such that $P^TAP = A$. For reasons explained further on, graphs whose adjacency matrices have this property are especially difficult to analyze and any procedure which efficiently reduces the number of such permutations through appropriate labeling of the nodes of the graph, will greatly reduce the complexity of following computations. Labeling of duplicate nodes was found to be a particularly fast and simple way to achieve this end for many graphs. It should be noted that this labeling alone does not necessarily reduce the automorphism group to the identity.

A well-known necessary condition for isomorphism involves the comparison of spectra (set of eigenvalues) of the 0,1 adjacency matrices of two graphs. Much effort has been devoted to nonisomorphic pairs for which the 0,1 adjacency matrices have the same spectrum. Such pairs of graphs are called cospectral [5]. Many cospectral pairs of graphs are distinguishable, however, through comparison of the spectra generated from adjacency matrices with numbers different from 0,1 used to represent nonedges and edges [6]. It appears that the use of 0 and 1 in the computation of the spectra of graphs ignores much of their inherent structure. A greater amount of information about the graph can be discovered from the examination of the spectra of adjacency matrices with 1's representing no edges, and a variable x representing an edge. Due to the nature of the characteristic polynomial, no generality is lost by replacing x with either a transcendental number or certain large functions of the number of nodes in the graph. Unfortunately neither option is feasible for use with the computer since a transcendental number is truncated to a rational and a large value of x yields errors in computation small in comparison to x but large in comparison to 1. A compromise solution was developed that allows x to be a function that increases linearly with the order of the graph yet is essentially as effective in distinguishing nonisomorphic graphs as is the more general procedure.

In order to utilize the information stored in the updated labels of the nodes, the diagonal elements of the adjacency matrices are assigned the value of the appropriate label. The eigenvalues of each modified adjacency matrix are then computed and compared. Should the resulting spectra not be identical, the graphs are not isomorphic. The preparation of the adjacency matrices for eigenvalue computation is effected in SUBROUTINE PREPAR and takes $0(n^2)$ time. The computation of the eigenvalues as well as the eigenvectors (for future use) of each matrix is carried out in SUBROUTINE SPECTR and takes $0(n^3)$ time.

Since two graphs that are not distinguished by the preceding sequence of tests may well be isomorphic, it is reasonable at this point to attempt to generate a permutation that relates the two graphs. The number of such possible permutations can be greatly limited with efficient use of the information stored in the eigenvectors of the two adjacency matrices. If A and B are two symmetric matrices with identical spectra then there exist orthogonal matrices U and V and a diagonal matrix D such that

$$U^TAU = D \qquad \text{and} \qquad V^TBV = D.$$

Further assuming that A and B are permutationally equivalent we have $B = P^TAP$ where P is a permutation matrix. Then:

$$U^TAU = V^TP^TAPV$$

$$D = V^TP^TUDU^TPV$$

$$U^TPVD = DU^TPV$$

and thus D commutes with $E = U^TPV$.

If $E = (e_{ij})$ and $D = (d_{ij})$ with $d_{ij} = 0$ for $i \neq j$, then $DE = (d_{ii}e_{ij})$ and $ED = (d_{jj}e_{ij})$. Thus if $d_{ii} \neq d_{jj}$, then $d_{jj}e_{ij} = d_{ii}e_{ij} \rightarrow e_{ij} = 0$.

Thus $E =$ $\begin{pmatrix} E_1 & & & 0 \\ & E_2 & & \\ & & \cdot & \\ 0 & & & \cdot \\ & & & E_m \end{pmatrix}$

where E_i is an orthogonal matrix of degree equal to the multiplicity of the ith distinct eigenvalue of A.

The relation $UE = PV$ serves to greatly restrict the number of permutations, P. Consider first the restrictions imposed by the existence of eigenvalues of multiplicity one. In this case, the corresponding matrix E_i is trivially (1) or (−1). Thus for each j such that d_{jj} is unique, we know that the set of permutations of

$$\begin{pmatrix} u_{1j} \\ u_{2j} \\ \cdot \\ \cdot \\ \cdot \\ u_{nj} \end{pmatrix} \text{ into } \begin{pmatrix} v_{1j} \\ v_{2j} \\ \cdot \\ \cdot \\ \cdot \\ v_{nj} \end{pmatrix} \text{ combined with}$$

$$\text{those of } \begin{pmatrix} -u_{1j} \\ -u_{2j} \\ \cdot \\ \cdot \\ \cdot \\ -u_{nj} \end{pmatrix} \text{ into } \begin{pmatrix} v_{1j} \\ v_{2j} \\ \cdot \\ \cdot \\ \cdot \\ v_{nj} \end{pmatrix} \text{ include}$$

all permutations of A into B.

$$\begin{pmatrix} u_{1j} \\ u_{2j} \\ \cdot \\ \cdot \\ \cdot \\ u_{nj} \end{pmatrix} \text{ and } \begin{pmatrix} v_{1j} \\ v_{2j} \\ \cdot \\ \cdot \\ \cdot \\ v_{nj} \end{pmatrix} \text{ are simply the}$$

eigenvectors of A and B corresponding to the eigenvalue d_{jj} and are easily computed.

Since the components of the eigenvectors of most modified adjacency matrices are unique, the number of possible permutations of A into B is small. In these cases, all possible permutations are quickly generated and checked until one is found that relates A and B or until the supply is exhausted. In the latter case, the graphs are not isomorphic.

As was mentioned earlier graphs that yield an adjacency matrix, A, and several permutations, P, such that $P^TAP = A$ are very difficult to work with. The reason for this phenomenon now becomes apparent. If the number of permutations of A into itself is large, then there will be many pairs of k and l such that $u_{kj} = u_{lj}$ for every j such that d_{jj} is unique. This, in turn, greatly increases the number of permutations

$$\text{of} \quad \begin{pmatrix} u_{1j} \\ u_{2j} \\ \cdot \\ \cdot \\ \cdot \\ u_{nj} \end{pmatrix} \quad \text{into} \quad \begin{pmatrix} v_{1j} \\ v_{2j} \\ \cdot \\ \cdot \\ \cdot \\ v_{nj} \end{pmatrix} \quad \text{and thus}$$

the amount of time which may be required to find a permutation of A into B and certainly the amount of time required to exhaust all possibilities is greatly increased. As an example, certain stochastic adjacency matrices have only one eigenvalue of multiplicity one. In many of these cases, the corresponding eigenvector components are all identical. In this, as well as in other less severe cases, the above described method of finding a permutation of A into B is not practical.

For such graphs a more sophisticated, as well as time consuming, approach is required. Again referring to the relation $UE = PV$, it is useful to examine the case when an eigenvalue has multiplicity two. In other words, assume that $d_{jj} = d_{kk}$ for $k = j, j+1$ but $d_{jj} \neq d_{kk}$ for all other k. Then we know that

$$\begin{pmatrix} u_{1j} & u_{1j+1} \\ u_{2j} & u_{2j+1} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ u_{nj} & u_{nj+1} \end{pmatrix} \begin{pmatrix} a & -b \\ b & a \end{pmatrix} = P \begin{pmatrix} v_{1j} & v_{1j+1} \\ v_{2j} & v_{2j+1} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ v_{nj} & v_{nj+1} \end{pmatrix}$$

or that

$$\begin{pmatrix} u_{1j} & u_{1j+1} \\ u_{2j} & u_{2j+1} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ u_{nj} & u_{nj+1} \end{pmatrix} \begin{pmatrix} a & b \\ b & -a \end{pmatrix} = P \begin{pmatrix} v_{1j} & v_{1j+1} \\ v_{2j} & v_{2j+1} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ v_{nj} & v_{nj+1} \end{pmatrix}$$

In each case, there are n possible values for a. Selecting i such that either $u_{ij} \neq 0$ or $u_{ij+1} \neq 0$ and assuming that there is a permutation of A into B which takes the ith node of A into the kth node of B, we have

$$\begin{aligned} u_{ij}a + u_{ij+1}b &= v_{kj} \\ -u_{ij}b + u_{ij+1}a &= v_{kj+1} \end{aligned}$$

or

$$\begin{aligned} u_{ij}a + u_{ij+1}b &= v_{kj} \\ u_{ij}b - u_{ij+1}a &= v_{kj+1} \end{aligned}$$

which yields

$$\begin{pmatrix} u_{ij} & u_{ij+1} \\ u_{ij+1} & -u_{ij} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} v_{kj} \\ v_{kj+1} \end{pmatrix}$$

or

$$\begin{pmatrix} u_{ij} & u_{ij+1} \\ -u_{ij+1} & u_{ij} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} v_{kj} \\ v_{kj+1} \end{pmatrix}.$$

Solving for a and b yields:

$$a = \frac{u_{ij}\, v_{kj} + u_{ij+1}\, v_{kj+1}}{u_{ij}^2 + u_{ij+1}^2},$$

$$b = \frac{-u_{ij}\, v_{kj+1} + u_{ij+1}\, v_{kj}}{u_{ij}^2 + u_{ij+1}^2}$$

or

$$a = \frac{u_{ij}\, v_{kj} - u_{ij+1}\, v_{kj+1}}{u_{ij}^2 + u_{ij+1}^2} ,$$

$$b = \frac{u_{ij}\, v_{kj+1} + u_{ij+1}\, v_{kj}}{u_{ij}^2 + u_{ij+1}^2} .$$

Note that $a^2 + b^2 = 1$ so that $|a| \leq 1$. Thus the permutations P are limited to those that satisfy

$$a \begin{bmatrix} u_{1j} \\ u_{2j} \\ \cdot \\ \cdot \\ \cdot \\ u_{nj} \end{bmatrix} + b \begin{bmatrix} u_{1j+1} \\ u_{2j+1} \\ \cdot \\ \cdot \\ \cdot \\ u_{nj+1} \end{bmatrix} = P \begin{bmatrix} v_{1j} \\ v_{2j} \\ \cdot \\ \cdot \\ \cdot \\ v_{nj} \end{bmatrix}$$

for any of the $2n$ pairs of (a,b), two pairs corresponding to each value of k. For each (a,b), the problem then reduces to the previously examined case of eigenvalues of multiplicity one. The only difference is that the process must be repeated $2n$ times before all possible permutations have been checked.

SUBROUTINE OPTEIG selects an optimal eigenvector, one that has as few multiple components as possible, and takes $0(n^3)$ time. If the selected eigenvector corresponds to an eigenvalue of multiplicity one, SUBROUTINE SINGLE is called upon. This subroutine is $0(n^3)$ but the leading coefficient can be very large if there are large numbers of multiple eigenvector components. SUBROUTINE DOUBLE calls SUBROUTINE SINGLE at most $2n$ times and thus is $0(n^4)$ with the possibility of a very large leading coefficient.

Due to the complex nature of the relation $UE = PV$ for blocks of E of dimension greater than two, little progress was made in obtaining meaningful data from the eigenvectors corresponding to eigenvalues of multiplicity greater than two.

Should neither the eigenvectors corresponding to eigenvalues of multiplicity one nor those of multiplicity two sufficiently restrict the number of permutations possible, still other algebraic approaches can be employed.

One such approach is the examination of the spectra of the n subgraphs of order $n - 1$ of the original graph. An obvious necessary condition for isomorphism is that the two graphs generate the same sets of spectra. Further there is the possibility that in a given set of spectra, many of them will be unique, thus creating the possibility of generating and testing all possible permutations in a small amount of time. Distinguishing node duplicates again helps to reduce the maximum number of permutations possible. Experience with a limited number of large stochastic matrices has shown, however, that this procedure is far more effective as a nonisomorphism test than as one that attempts to generate and test all possible permutations. This procedure is contained in SUBROUTINE SUBVAL and takes $0(n^4)$ time.

Carrying the eigenvector analysis further all possible permutations can be derived through examination of eigenvectors of subgraphs of the original graph. This method has proved to be one of the most successful in distinguishing very difficult large, stochastic graphs. For simplicity and speed, all submatrices of order $n - 1$ are searched for the optimum or a sufficiently satisfactory eigenvector corresponding to an eigenvalue of multiplicity one. This eigenvector is then compared with the corresponding eigenvector generated from each of the subgraphs of the second graph that has a spectrum matching the spectrum of the subgraph which generated the optimal eigenvector. This procedure is executed in SUBROUTINE SUBVEC. The order is $0(n^4)$ as SUBROUTINE SINGLE can be called a maximum of n times. This approach tends to be more time consuming than the others mentioned since it could require the generation of n eigenvector matrices.

In a similar fashion, one could examine the eigenvectors of subgraphs corresponding to eigenvalues of multiplicity two. Another possible approach would be to examine the n^2 spectra generated by sequentially generating all possible subgraphs of order $n - 2$. These and other possible procedures are $0(n^5)$ or worse and very time consuming. It does not appear that the trade off of speed for effectiveness dictates inclusion of such unwieldly approaches in the algorithm.

3. Computer Implementation

The algorithm was organized into fourteen subroutines and coded in FORTRAN V for use with the UNIVAC 1108 computer. SUBROUTINE GISOM serves as the coordinating subroutine and is the

only subroutine that need be referenced by a main program. Each subroutine listing is fully commented, indicating values of variables on input, output and during the life of the subroutine.

All eigenvalues and eigenvectors were calculated using the EISPACK software described in [9]. Reference to [9] should be made for a detailed explanation of the function, running time, error messages and accuracy of the EISPACK subroutines. Throughout the program two numbers, A and B are considered identical if $||A| - |B|| <$ MACHP * $(1 + |A|)$. The precision constant, MACHP, is determined and set by the user in the call to GISOM. For use with the EISPACK software on the UNIVAC 1108, a value of MACHP $= 0.0003$ was found to be satisfactory.

Another user option in the call to GISOM command sets a limit on the number of permutations that may be tested for a given pair of graphs. Generally, this value, MAXT, can be set to a very large number, say 5000, since very little time is expended on each test. MAXT is of greatest use when it is desired to sequentially test several pairs of very large matrices without spending too much time on any one pair.

Time rather than storage conservation was emphasized. The subroutines utilize five $n \times n$ arrays and several single dimension $0(n)$ arrays, all of which are passed as part of the subroutine call statements. There are no common blocks and no input-output statements in the subroutine as all pertinent data is returned through the CALL GISOM command in the main program. The STATUS and ERR variables serve as the key to what happened when the two graphs were tested for isomorphism. The other variables contain a variety of data whose meaning depends on the value of ERR and STATUS. This data can often be quite useful in determining exactly why two graphs are not isomorphic. The interpretation of ERR and STATUS is given with the commentary of each subroutine. A list of the program follows.

```
      SUBROUTINE GISOM (A,B,C,XA,XB,DA,DB,EA,EB,FA,FB,SC,SD,SE,CE,CF,G,H
     +                  ,P,STATUS,ERR,MACHP,MAXT,N,NM,NT)
C
C  THIS SUBROUTINE IS THE CONTROL SUBROUTINE FOR THE ALGORITHM.
C
C  ON INPUT:
C     A AND B CONTAIN THE 0,1 FORM OF THE ADJACENCY MATRICES OF THE TWO
C        GRAPHS TO BE COMPARED.  THE DIAGONAL ELEMENTS CONTAIN LABELS
C        WHICH MAY RANGE IN VALUE FROM 0 TO 99.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.  A
C        VALUE OF .0003 IS SUGGESTED WHEN THE PROGRAM IS RUN ON A
C        MACHINE COMPARABLE TO THE UNIVAC 1108 AND WITH SOFTWARE
C        EQUIVALENT TO EISPACK IN ACCURACY.
C     MAXT IS APPROXIMATELY THE MAXIMUM NUMBER OF PERMUTATIONS THAT MAY
C        BE TESTED.  EXCEPT IN CASES OF SEVERE TIME RESTRICTIONS, A
C        VALUE OF 5000 IS SUGGESTED.
C     N IS THE ORDER OF THE GRAPHS TO BE COMPARED.
C     NM IS THE DIMENSION OF MOST OF THE ARRAYS.
C     NT = 2*NM AND IS THE DIMENSION OF CE AND CF.
C
C  ON RETURN:
C     A AND B ARE POSSIBLY DESTROYED.
C     P CONTAINS A PERMUTATION SUCH THAT A(I,J)=B(P(I),P(J)) FOR ALL I,J
C        SHOULD SUCH A PERMUTATION BE FOUND TO EXIST.
C     STATUS RANGES IN VALUE FROM 1 TO 10 AND RECORDS THE POINT IN GISOM
C        WHERE THE ALGORITHM TERMINATED.  FOR EXAMPLE, A VALUE OF STATUS
C        =3 MEANS THAT THE ALGORITHM TERMINATED FOLLOWING THE NODE
C        VALENCE CHECK WHICH IMPLIES THAT THE NODE VALENCES OF THE TWO
C        GRAPHS DIFFERED AND THUS THAT THE TWO GRAPHS ARE NOT
C        ISOMORPHIC.
C     ERR CONTAINS INFORMATION PERTAINING TO THE REASON FOR THE
C        TERMINATION OF THE ALGORITHM.  AN EXPLANATION OF ITS MEANING
C        CAN BE FOUND WITH THE COMMENTS IN THE SUBROUTINE THAT WAS LAST
C        CALLED BY GISOM.  FOR EXAMPLE, A STATUS VALUE OF 7 COMBINED
C        WITH AN ERR OF 0 IMPLIES THAT A PERMUTATION WAS FOUND THAT
C        RELATES A AND B THROUGH USE OF AN EIGENVECTOR CORRESPONDING TO
C        AN EIGENVALUE OF MULTIPLICITY ONE.
C     MACHP, MAXT, N, NM, AND NT ARE UNCHANGED.
C
C  ALL OTHER VARIABLES ARE CLASSIFIED AS WORKING VARIABLES.  SUCH
C  VARIABLES MAY CONTAIN RELEVANT INFORMATION DEPENDING ON THE VALUES OF
C  STATUS AND ERR.
C
      INTEGER N,NM,NT,ERR,MAXT,G(NM),H(NM),P(NM),STATUS,JA
      REAL A(NM,NM),B(NM,NM),C(NM,NM),XA(NM,NM),XB(NM,NM),DA(NM),DB(NM),
     +     EA(NM),EB(NM),FA(NM),FB(NM),SC(NM),SD(NM),SE(NM),CE(NT),MACHP
     +     ,CF(NT)
C
      ERR=0
      STATUS=1
      IF (N.GT.NM) RETURN
C
C  COMPARE THE NODE LABELS OF THE TWO GRAPHS.
C
      STATUS=2
      CALL LABEL (A,B,DA,DB,SC,ERR,N,NM)
      IF (ERR.GT.0) RETURN
C
C  COMPARE THE NODE VALENCES AND UPDATE THE CORRESPONDING NODE LABELS.
```

```
C
      STATUS=3
      CALL VALENC (A,B,DA,DB,SC,ERR,N,NM)
      IF (ERR.GT.0) RETURN
C
C  FIND, LABEL, AND COMPARE ORDINARY AND CONNECTED DUPLICATE NODES.
C
  10  STATUS=STATUS+1
         CALL DUPLIC (A,B,DA,DB,SC,SD,ERR,STATUS,N,NM)
         IF (ERR.GT.0) RETURN
      IF (STATUS.EQ.4) GO TO 10
C
C  CONVERT THE 0,1 FORM OF THE ADJACENCY MATRIX TO THE 1,X FORM.
C
      CALL PREPAR (A,B,DA,DB,N,NM)
C
C  COMPUTE AND COMPARE THE SPECTRA OF THE TWO ADJACENCY MATRICES.
C
      STATUS=6
      CALL SPECTR (A,B,XA,XB,EA,EB,SC,G,ERR,MACHP,N,NM)
      IF (ERR.NE.0) RETURN
C
C  LOCATE A SUITABLE OR OPTIMAL EIGENVECTOR.
C
      CALL OPTEIG (XB,DB,G,ERR,MACHP,JA,N,NM)
      GO TO (20,30,50), ERR
C
C  ATTEMPT TO FIND A PERMUTATION THAT RELATES A AND B THROUGH USE OF AN
C  EIGENVECTOR THAT CORRESPONDS TO AN EIGENVALUE OF MULTIPLICITY ONE.
C
  20  STATUS=7
      CALL SINGLE (A,B,XA,XB,FA,FB,H,P,ERR,MACHP,MAXT,JA,N,NM)
      GO TO 40
C
C  ATTEMPT TO FIND A PERMUTATION THAT RELATES A AND B THROUGH USE OF AN
C  EIGENVECTOR THAT CORRESPONDS TO AN EIGENVALUE OF MULTIPLICITY TWO.
C
  30  STATUS=8
      CALL DOUBLE (A,B,C,XA,XB,CE,CF,FA,FB,H,P,ERR,MAXT,MACHP,JA,N,NM,NT
     +             )
  40  IF (ERR.NE.4) RETURN
C
C  COMPUTE AND COMPARE THE SPECTRA OF THE N ORDER N-1 SUBGRAPHS OF THE
C  ORIGINAL GRAPHS.
C
  50  STATUS=9
      CALL SUBVAL (A,B,C,XA,SC,SD,SE,H,P,ERR,MACHP,MAXT,N,NM)
      IF (ERR.NE.0.AND.ERR.NE.5+N) RETURN
C
C  ATTEMPT TO FIND A PERMUTATION THAT RELATES A AND B THROUGH USE OF AN
C  EIGENVECTOR THAT CORRESPONDS TO AN EIGENVALUE OF MULTIPLICITY ONE IN
C  A SUBGRAPH OF ORDER N-1.
C
      STATUS=10
      CALL SUBVEC (A,B,C,XA,XB,DA,DB,EA,EB,FA,FB,SC,SD,SE,G,H,P,ERR,
     +             MACHP,MAXT,N,NM)
C
      RETURN
      END
```

```
      SUBROUTINE LABEL (A,B,DA,DB,SC,ERR,N,NM)
C
C  THIS SUBROUTINE RECORDS AND COMPARES THE INITIAL LABELS OF THE NODES
C  OF THE TWO GRAPHS TO BE COMPARED.
C
C  ON INPUT:
C     A AND B CONTAIN THE TWO ADJACENCY MATRICES TO BE COMPARED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A AND B ARE UNCHANGED.
C     DA AND DB CONTAIN THE DIAGONAL ELEMENTS -LABELS- OF A AND B.
C     ERR = 0 IF DA AND DB ARE ISOMORPHIC.  OTHERWISE ERR IS THE INDEX
C        OF THE ELEMENT OF DA WHICH CANNOT BE MATCHED IN DB.
C     N AND NM ARE UNCHANGED.
C
C  WORKING ARRAYS:   SC
C
      INTEGER N,NM,ERR,I
      REAL A(NM,NM),B(NM,NM),DA(NM),DB(NM),SC(NM)
C
C  RECORD THE LABELS.
C
      DO 10 I=1,N
         DA(I)=A(I,I)
         DB(I)=B(I,I)
   10 CONTINUE
C
C  COMPARE THE LABELS.
C  NOTE THAT SINCE THE LABELS ARE INTEGERS, THE MACHINE PRECISION
C  CONSTANT IS ZERO.
C
      CALL COMP (DA,DB,SC,ERR,0,N,NM)
C
      RETURN
      END
```

```
      SUBROUTINE COMP (RA,RB,RC,ERR,MACHP,N,NM)
C
C  THIS SUBROUTINE DETERMINES WHETHER OR NOT TWO VECTOR ARRAYS ARE
C  ISOMORPHIC.
C
C  ON INPUT:
C     RA AND RB ARE THE TWO VECTOR ARRAYS TO BE COMPARED.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     RA AND RB ARE UNCHANGED.
C     ERR = 0 IF RA AND RB ARE ISOMORPHIC.  OTHERWISE ERR IS THE INDEX
C        OF THE ELEMENT OF RA WHICH CANNOT BE MATCHED IN RB.
C     MACHP, N, AND NM ARE UNCHANGED.
C
C  WORKING ARRAYS:   RC
C
      INTEGER NM,N,ERR,I,J
      REAL RA(NM),RB(NM),RC(NM),MACHP,VAL
C
      VAL=1
      ERR=0
C
C  COPY RB INTO RC AND COMPUTE A RELATIVELY LARGE NUMBER.
C
      DO 10 I=1,N
         RC(I)=RB(I)
         VAL=VAL+ABS(RB(I))
   10 CONTINUE
C
C  COMPARE THE TWO ARRAYS.
C
      DO 30 I=1,N
         DO 20 J=1,N
            IF (ABS(RA(I)-RC(J)).GT.MACHP*(1+ABS(RC(J)))) GO TO 20
               RC(J)=VAL
               GO TO 30
   20       CONTINUE
            ERR=I
            RETURN
   30 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE VALENC (A,B,DA,DB,SC,ERR,N,NM)
C
C  THIS SUBROUTINE COMPUTES AND COMPARES THE NODE VALENCES OF THE
C  TWO GRAPHS.
C
C  ON INPUT:
C     A AND B CONTAIN THE TWO ADJACENCY MATRICES TO BE COMPARED.
C     DA AND DB CONTAIN THE NODE LABELS OF A AND B.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A AND B ARE UNCHANGED.
C     DA AND DB CONTAIN THE UPDATED NODE LABELS.
C     ERR = 0 IF DA AND DB ARE ISOMORPHIC.  OTHERWISE ERR IS THE INDEX
C        OF THE ELEMENT OF DA WHICH CANNOT BE MATCHED IN DB.
C     N AND NM ARE UNCHANGED.
C
C  WORKING ARRAYS:   SC
C
      INTEGER NM,N,ERR,I,J
      REAL A(NM,NM),B(NM,NM),DA(NM),DB(NM),SC(NM)
C
      DO 20 I=1,N
C
C  MAKE ROOM FOR NEW INFORMATION.
C
         DA(I)=100*DA(I)
         DB(I)=100*DB(I)
C
C  COMPUTE AND RECORD NODE VALENCES - UPDATE LABELS.
C
         DO 10 J=1,N
            IF (I.EQ.J) GO TO 10
               DA(I)=DA(I)+A(I,J)
               DB(I)=DB(I)+B(I,J)
   10    CONTINUE
   20 CONTINUE
C
C  COMPARE UPDATED LABELS.
C
      CALL COMP (DA,DB,SC,ERR,0,N,NM)
C
      RETURN
      END
```

```
      SUBROUTINE DUPLIC (A,B,DA,DB,SC,SD,ERR,Q,N,NM)
C
C  THIS SUBROUTINE LOCATES AND DISTINGUISHES DUPLICATE NODES THROUGH
C  LABEL MODIFICATION.
C
C  ON INPUT:
C     A AND B CONTAIN THE TWO ADJACENCY MATRICES TO BE COMPARED.
C     DA AND DB CONTAIN CURRENT NODE LABEL DATA.
C     Q = 4 IF 0'S ARE TO BE PLACED ON THE DIAGONAL OF A AND B, AND
C        Q = 5 IF 1'S ARE TO BE USED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A AND B HAVE DIAGONAL ENTRIES OF ALL 0'S OR ALL 1'S BUT ARE
C        OTHERWISE UNCHANGED.
C     DA AND DB CONTAIN UPDATED LABEL DATA.
C     ERR = 0 IF DA AND DB ARE ISOMORPHIC.  OTHERWISE ERR IS THE INDEX
C        OF THE ELEMENT OF DA WHICH CANNOT BE MATCHED IN DB.
C     Q, N, AND NM ARE UNCHANGED.
C
C  WORKING VARIABLES:   SC,SD
C
      INTEGER NM,N,NN,I,J,K,L,ERR,Q
      REAL A(NM,NM),B(NM,NM),DA(NM),DB(NM),SC(NM),SD(NM)
C
      NN=N-1
C
      DO 10 I=1,N
C
C  SET DIAGONAL ELEMENTS TO 0 OR 1.
C
         A(I,I)=Q-4
         B(I,I)=Q-4
C
C  MAKE ROOM FOR NEW DATA.
C
         DA(I)=100*DA(I)
         DB(I)=100*DB(I)
C
C  STORE DA IN SC AND DB IN SD.
C
         SC(I)=DA(I)
         SD(I)=DB(I)
  10  CONTINUE
C
      DO 60 I=1,NN
         L=I+1
         DO 50 K=L,N
C
C  SEARCH FOR IDENTICAL ROWS IN A.
C
            IF (SC(I).NE.SC(K)) GO TO 30
               DO 20 J=1,N
                  IF (A(I,J).NE.A(K,J)) GO TO 30
  20           CONTINUE
                  DA(K)=DA(K)+1
                  DA(I)=DA(I)+2
```

```
C
C  SEARCH FOR IDENTICAL ROWS IN B.
C
  30         IF (SD(I).NE.SD(K)) GO TO 50
                DO 40 J=1,N
                   IF (B(I,J).NE.B(K,J)) GO TO 50
  40            CONTINUE
                   DB(K)=DB(K)+1
                   DB(I)=DB(I)+2
  50      CONTINUE
  60   CONTINUE
C
C  COMPARE UPDATED LABELS.
C
       CALL COMP (DA,DB,SC,ERR,0,N,NM)
C
       RETURN
       END
```

```
      SUBROUTINE PREPAR (A,B,DA,DB,N,NM)
C
C  THIS SUBROUTINE CONVERTS THE ADJACENCY MATRIX FROM 0,1 FORMAT TO 1,X
C  FORMAT.  IN ADDITION THE LABELS ARE CONVERTED INTO MULTIPLES OF N
C  AND ARE ENTERED AS DIAGONAL ELEMENTS OF A AND B.
C
C  ON INPUT:
C     A AND B ARE THE ADJACENCY MATRICES IN 0,1 FORMAT.
C     DA AND DB CONTAIN CURRENT NODE LABEL DATA.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A AND B CONTAIN THE TWO ADJACENCY MATRICES IN 1,X FORM WITH
C        MODIFIED DIAGONAL ELEMENTS IN MULTIPLES OF N.
C     DA AND DB CONTAIN THE NODE LABEL DATA IN THE FORM OF MULTIPLES
C        OF N.
C     N AND NM ARE UNCHANGED.
C
      INTEGER NM,N,L
      REAL A(NM,NM),B(NM,NM),DA(NM),DB(NM),FN,NA,NB,QA,QB,INF
C
C  FN IS THE FUNCTION FOR X-1 IN THE 1,X FORM OF THE ADJACENCY MATRIX.
C
      FN=N/2
      QA=-1
      QB=-1
      NA=0
      NB=0
      INF=1000000000
C
C  CONVERT A AND B TO 1,X FORM.
C
      DO 20 I=1,N
         DO 10 J=1,N
            A(I,J)=A(I,J)*FN+1
            B(I,J)=B(I,J)*FN+1
   10    CONTINUE
   20 CONTINUE
C
      DO 50 I=1,N
C
C  LABEL DIAGONAL ELEMENTS OF A IN MULTIPLES OF N.
C
         L=1
         DO 30 J=1,N
            IF (DA(J).LT.DA(L)) L=J
   30    CONTINUE
         IF (DA(L).GT.QA) NA=NA+N
         A(L,L)=NA
         QA=DA(L)
         DA(L)=INF
C
C  LABEL DIAGONAL ELEMENTS OF B IN MULTIPLES OF N.
C
         L=1
         DO 40 J=1,N
            IF (DB(J).LT.DB(L)) L=J
```

```
   40      CONTINUE
           IF (DB(L).GT.QB) NB=NB+N
           B(L,L)=NB
           QB=DB(L)
           DB(L)=INF
   50   CONTINUE
C
C  RESTORE DA AND DB.
C
        DO 60 I=1,N
           DA(I)=A(I,I)
           DB(I)=B(I,I)
   60   CONTINUE
C
        RETURN
        END
```

```
      SUBROUTINE SPECTR (A,B,XA,XB,EA,EB,SC,G,ERR,MACHP,N,NM)
C
C  THIS SUBROUTINE CALCULATES AND COMPARES THE SPECTRA OF THE TWO
C  MODIFIED ADJACENCY MATRICES.  IN ADDITION, DATA PERTAINING TO THE
C  MULTIPLICITY OF THE EIGENVALUES IS RECORDED.
C
C  ON INPUT:
C     A AND B CONTAIN THE ADJACENCY MATRICES OF THE TWO GRAPHS.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A AND B ARE UNCHANGED.
C     XA AND XB CONTAIN THE EIGENVECTORS OF A AND B IN THE SAME ORDER
C        AS THE CORRESPONDING EIGENVALUES.
C     EA AND EB CONTAIN THE EIGENVALUES OF A AND B IN NONDECREASING
C        ORDER.
C     G OF I IS 1 IF THE I TH AND I+1 TH EIGENVALUES OF A ARE IDENTICAL
C        AND IS 0 OTHERWISE.
C     ERR IS NEGATIVE IF AN ERROR CONDITION WAS RAISED IN THE EISPACK
C        SOFTWARE.  ERR IS 0 IF THE SPECTRA ARE IDENTICAL.  OTHERWISE,
C        ERR IS THE INDEX OF THE EIGENVALUE OF A THAT COULD NOT BE
C        MATCHED WITH AN EIGENVALUE OF B.
C     MACHP, N, AND NM ARE UNCHANGED.
C
C  WORKING VARIABLES:   SC
C
      INTEGER NM,N,ERR,I,NN,G(NM)
      REAL A(NM,NM),B(NM,NM),XA(NM,NM),XB(NM,NM),EA(NM),EB(NM),SC(NM),
     +     MACHP
C
      NN=N-1
C
C  FIND THE EIGENVALUES AND EIGENVECTORS OF A.
C
      CALL TRED2 (NM,N,A,EA,SC,XA)
      CALL TQL2 (NM,N,EA,SC,XA,ERR)
      IF (ERR.NE.0) GO TO 10
C
C  FIND THE EIGENVALUES AND EIGENVECTORS OF B.
C
         CALL TRED2 (NM,N,B,EB,SC,XB)
         CALL TQL2 (NM,N,EB,SC,XB,ERR)
  10  ERR=-ERR
      IF (ERR.EQ.0) CALL COMP (EA,EB,SC,ERR,MACHP,N,NM)
C
C  RECORD EIGENVALUE MULTIPLICITY DATA.
C
      IF (ERR.NE.0) RETURN
      DO 20 I=1,NN
         G(I)=0
         IF (ABS(EA(I)-EA(I+1)).LT.MACHP*(1+ABS(EA(I)))) G(I)=1
  20  CONTINUE
      G(N)=0
C
      RETURN
      END
```

```
      SUBROUTINE OPTEIG (XB,DB,G,ERR,MACHP,JA,N,NM)
C
C  THIS SUBROUTINE SELECTS AN OPTIMAL EIGENVECTOR FOR USE IN GENERATING
C  A SERIES OF PERMUTATIONS.
C
C  ON INPUT:
C     XB IS THE EIGENVECTOR MATRIX OF B.
C     DB CONTAINS THE DIAGONAL ELEMENTS OF B.
C     G CONTAINS EIGENVALUE MULTIPLICITY INFORMATION.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     XB IS ESSENTIALLY UNCHANGED.  THE POSSIBILITY EXISTS THAT ANY TWO
C        EIGENVECTORS CORRESPONDING TO THE SAME EIGENVALUE COULD BE
C        INTERCHANGED.
C     DB AND G ARE UNCHANGED.
C     ERR = 1 IF THE EIGENVECTOR SELECTED CORRESPONDS TO AN EIGENVALUE
C        OF MULTIPLICITY 1.  ERR=2 IF THE CORRESPONDING EIGENVALUE HAS
C        MULTIPLICITY 2.  ERR=3 IF NO SUITABLE EIGENVECTOR COULD BE
C        FOUND.
C     MACHP IS UNCHANGED.
C     JA IS THE COLUMN INDEX OF THE OPTIMAL EIGENVECTOR, IF ONE EXISTS.
C     N AND NM ARE UNCHANGED.
C
      INTEGER NM,N,I,J,PN,LT,PX,ERR,JA,LI,LM,L,NN,G(NM)
      REAL XB(NM,NM),DB(NM),MACHP
C
      NN=N-1
      ERR=3
      PN=1000
C
C  ALL EIGENVECTORS WITH THREE OR MORE IDENTICAL COMPONENTS OR THAT
C  CORRESPOND TO EIGENVALUES OF MULTIPLICITY THREE OR MORE ARE
C  ELIMINATED FROM CONSIDERATION.  EIGENVECTORS THAT CORRESPOND TO
C  EIGENVALUES OF MULTIPLICITY ONE ARE CONSIDERED PREFERENTIALLY TO
C  THOSE WHICH HAVE MULTIPLICITY TWO.  EIGENVECTORS WHICH HAVE FEW PAIRS
C  OF IDENTICAL COMPONENTS ARE FAVORED OVER THOSE WITH MANY SUCH PAIRS.
C
      DO 80 J=1,N
         LT=0
         PX=1
C
C  CHECK FOR MULTIPLICITY OF ASSOCIATED EIGENVALUE.
C
         IF (J.EQ.1) GO TO 20
            IF (G(J-1).EQ.0) GO TO 20
               IF (J.EQ.2) GO TO 10
                  IF (G(J-2).EQ.1.) GO TO 80
   10             IF (G(J).EQ.1) GO TO 80
                     LT=4
                     PX=4
                     GO TO 30
   20    IF (G(J).EQ.0) GO TO 30
            IF (G(J+1).EQ.1) GO TO 80
               LT=4
               PX=2
```

```
C
C  CHECK FOR IDENTICAL COMPONENTS.
C
  30      DO 50 I=1,NN
             LI=I+1
             LM=0
             DO 40 L=LI,N
                IF (ABS(XB(I,J)-XB(L,J)+DB(I)-DB(L)).GT.MACHP) GO TO 40
                   LT=LT+1
                   IF (LM.EQ.1) GO TO 80
                      LM=1
  40            CONTINUE
  50         CONTINUE
             IF (LT.GE.PN) GO TO 80
                JA=J
                ERR=PX
                IF (ERR.LT.4) GO TO 70
                   ERR=2
                   JA=JA-1
C
C  INTERCHANGE TWO EIGENVECTORS THAT CORRESPOND TO THE SAME EIGENVALUE
C  SO THAT IN THE RESULTING PAIR THE FIRST WILL BE MORE SUITABLE THAN
C  THE LAST.
C
                   DO 60 I=1,N
                      TEMP=XB(I,JA)
                      XB(I,JA)=XB(I,JA+1)
                      XB(I,JA+1)=TEMP
  60                  CONTINUE
  70             PN=LT
C
C  IF EIGENVECTOR IS SUITABLE -NOT NECESSARILY OPTIMAL- RETURN.
C
                IF (LT.LT.4) RETURN
  80   CONTINUE
C
       RETURN
       END
```

```
      SUBROUTINE SINGLE (A,B,XA,XB,FA,FB,H,P,ERR,MACHP,MAXT,JA,N,NM)
C
C  THIS SUBROUTINE GENERATES AND TESTS UP TO MAXT PERMUTATIONS FROM DATA
C  FOUND IN AN EIGENVECTOR CORRESPONDING TO AN EIGENVALUE OF
C  MULTIPLICITY ONE.
C
C  ON INPUT:
C     A AND B CONTAIN THE ADJACENCY MATRICES OF THE TWO GRAPHS.
C     XA AND XB ARE THE EIGENVECTOR MATRICES OF A AND B THOUGH ONLY THE
C        JA TH COLUMNS OF XA AND XB ARE USED IN THIS SUBROUTINE.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     MAXT IS THE MAXIMUM NUMBER OF PERMUTATIONS THAT CAN BE TESTED.
C     JA IS THE COLUMN OF XA AND XB WHICH CONTAINS THE EIGENVECTORS TO
C        BE EXAMINED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A, B, XA, AND XB ARE UNCHANGED.
C     P CONTAINS THE LAST PERMUTATION THAT WAS CHECKED AGAINST A AND B.
C        IF ERR=0, THIS PERMUTATION RELATES A AND B.
C     ERR = -1 IF NO SUITABLE PERMUTATIONS COULD BE GENERATED AND THE
C        GRAPHS ARE NOT ISOMORPHIC.  ERR=1 IF THREE IDENTICAL COMPONENTS
C        WERE FOUND IN THE EIGENVECTOR.  ERR=3 IF ALL POSSIBLE
C        PERMUTATIONS FAILED WHEN TESTED AGAINST A AND B AND THE GRAPHS
C        ARE NOT ISOMORPHIC.  ERR=4 IF THE MAXIMUM NUMBER OF
C        PERMUTATIONS WERE TRIED WITHOUT SUCCESS.  ERR=0 IF A
C        PERMUTATION WAS FOUND TO RELATE A AND B.
C     MACHP, MAXT, JA, N, AND NM ARE UNCHANGED.
C
C  WORKING VARIABLES:   FA,FB,H
C
      INTEGER NM,N,ERR,JA,I,M,H(NM),P(NM),LT,MAXT
      REAL A(NM,NM),B(NM,NM),XA(NM,NM),XB(NM,NM),FA(NM),FB(NM),MACHP
C
      M=0
C
C  USE THE NODE LABELS TO AIDE IN THE SEPARATION OF IDENTICAL
C  COMPONENTS.
C
      DO 10 I=1,N
         FA(I)=A(I,I)+XA(I,JA)
         FB(I)=B(I,I)+XB(I,JA)
   10 CONTINUE
C
C  GENERATE A POTENTIAL PERMUTATION AS WELL AS THE DATA NECESSARY TO
C  FIND ALL OTHER POSSIBLE PERMUTATIONS.
C
   20 CALL PERM (FA,FB,H,P,ERR,MACHP,LT,N,NM)
      IF (ERR.EQ.1) RETURN
         IF (ERR.EQ.-1) GO TO 30
C
C  TEST UP TO MAXT PERMUTATIONS AGAINST A AND B.
C
            CALL TEST (A,B,H,P,ERR,MAXT,LT,N,NM)
            IF (ERR.NE.3) RETURN
   30 IF (M.EQ.1) RETURN
C
```

```
C  SHOULD ALL PERMUTATIONS FAIL, REPEAT THE PROCESS WITH ONE EIGENVECTOR
C  NEGATED TO INCLUDE ALL POSSIBLE CASES.
C
         DO 40 I=1,N
            FB(I)=B(I,I)-XB(I,JA)
  40     CONTINUE
         M=1
         GO TO 20
C
      END
```

```
      SUBROUTINE PERM (FA,FB,H,P,ERR,MACHP,LT,N,NM)
C
C  THIS SUBROUTINE GENERATES A SINGLE PERMUTATION BETWEEN THE COMPONENTS
C  OF TWO VECTORS AND RECORDS THE DATA NECESSARY TO GENERATE ALL OTHER
C  POSSIBLE PERMUTATIONS BETWEEN THE COMPONENTS OF THE TWO VECTORS.  THE
C  VECTORS MUST NOT CONTAIN MORE THAN TWO IDENTICAL COMPONENTS.
C
C  ON INPUT:
C     FA AND FB ARE THE TWO VECTORS FROM WHICH ALL PERMUTATIONS WILL
C        BE GENERATED.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     FA IS UNCHANGED.
C     FB IS DESTROYED.
C     H CONTAINS THE INDICES OF THE PAIRS OF IDENTICAL COMPONENTS OF FA.
C     P CONTAINS A PERMUTATION OF THE COMPONENTS OF FA INTO THOSE OF FB.
C     ERR = -1 IF THE TWO VECTORS ARE FOUND TO BE NON-ISOMORPHIC.
C        ERR=1 IF THREE OR MORE COMPONENTS OF FA ARE IDENTICAL.
C        ERR=0 OTHERWISE.
C     MACHP IS UNCHANGED.
C     LT IS THE NUMBER OF IDENTICAL PAIRS OF COMPONENTS FOUND IN FA.
C     N AND NM ARE UNCHANGED.
C
      INTEGER NM,N,I,J,L,LT,H(NM),P(NM),ERR
      REAL FA(NM),FB(NM),MACHP,INF
C
      ERR=1
      LT=0
      INF=-10
C
      DO 70 I=1,N
         L=-1
C
C  FIND ALL COMPONENTS OF FB THAT MATCH THE CURRENT COMPONENT OF FA.
C
         DO 30 J=1,N
            IF (ABS(FA(I)-FB(J)).GT.MACHP) GO TO 30
               IF (L) 10,20,80
 10               FB(J)=INF
                  P(I)=J
 20            L=L+1
 30      CONTINUE
C
         IF (L) 40,70,50
 40         ERR=-1
            RETURN
C
C  RECORD THE INDEX OF THE FIRST HALF OF A IDENTICAL PAIR IN FA.
C
 50         LT=LT+1
            H(2*LT-1)=I
C
C  FIND AND RECORD THE INDEX OF THE SECOND HALF OF THE IDENTICAL PAIR
C  IN FA.
C
```

```
            DO 60 J=1,N
               IF (ABS(FA(I)-FA(J)).GT.MACHP.OR.I.EQ.J) GO TO 60
                  H(2*LT)=J
                  GO TO 70
   60       CONTINUE
   70 CONTINUE
C
      ERR=0
   80 RETURN
C
      END
```

```
      SUBROUTINE TEST (A,B,H,P,ERR,MAXT,LT,N,NM)
C
C  THIS SUBROUTINE TESTS UP TO MAXT PERMUTATIONS AGAINST THE ORIGINAL
C  GRAPHS.
C
C  ON INPUT:
C     A AND B CONTAIN THE ADJACENCY MATRICES OF THE TWO GRAPHS.
C     H CONTAINS THE PAIRS OF INDICES OF P WHICH MUST BE INTERCHANGED
C        BEFORE ALL POSSIBLE PERMUTATIONS CAN BEEN TESTED.
C     P CONTAINS THE FIRST PERMUTATION TO BE CHECKED.
C     MAXT IS THE MAXIMUM NUMBER OF PERMUTATIONS WHICH MAY BE TESTED.
C     LT IS THE NUMBER OF PAIRS OF INDICES OF P THAT MAY HAVE TO BE
C        INTERCHANGED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A, B, AND H ARE UNCHANGED.
C     P CONTAINS THE LAST PERMUTATION THAT WAS CHECKED AGAINST A AND B.
C        IF ERR=0, THEN THIS PERMUTATION RELATES A AND B.
C     ERR = 0 IF A PERMUTATION WAS SUCCESSFUL IN RELATING A AND B.
C        ERR=3 IF ALL PERMUTATIONS FAILED WHEN TESTED AGAINST A AND B.
C        ERR=4 IF MAXT PERMUTATIONS WERE TRIED WITHOUT SUCCESS.
C     LT, N, AND NM ARE UNCHANGED.
C
      INTEGER NM,N,LT,ERR,LB,I,J,BIN,MAXT,T,L,K,H(NM),P(NM),M
      REAL A(NM,NM),B(NM,NM)
C
C  LB IS THE MAXIMUM NUMBER LESS 1 OF PERMUTATIONS WHICH MAY HAVE TO BE
C  TESTED.
C
      LB=2**LT-1
C
C  BIN CONTROLS WHICH PAIRS OF INDICES ARE TO BE INTERCHANGED EACH TIME.
C
      BIN=0
      T=0
      ERR=0
C
  10  T=T+1
C
C  TEST THE PERMUTATION IN P AGAINST A AND B.
C
      DO 30 I=1,N
         L=P(I)
         DO 20 J=1,N
            K=P(J)
            IF (B(L,K).NE.A(I,J)) GO TO 40
  20     CONTINUE
  30  CONTINUE
      RETURN
C
C  CHECK IF ALL POSSIBLE PERMUTATIONS HAVE BEEN TRIED.
C
  40  IF (BIN.LT.LB) GO TO 50
         ERR=3
         RETURN
C
```

```
C  CHECK IF MAXT PERMUTATIONS HAVE BEEN TRIED.
C
  50  IF (T.LT.MAXT) GO TO 60
         ERR=4
         RETURN
C
C  GENERATE THE NEXT PERMUTATION TO BE TESTED.
C
  60  BIN=BIN+1
C
C  SEQUENTIALLY INTERCHANGE I+1 PAIRS OF VALUES OF P WHERE 2**I IS THE
C  LARGEST POWER OF 2 THAT DIVIDES BIN.  ON THE AVERAGE, TWO PAIRS WILL
C  BE INTERCHANGED EACH TIME, INDEPENDENT OF THE VALUE OF LT.
C
      M=1
      DO 70 I=1,LT
         L=H(2*I-1)
         K=H(2*I)
         J=P(L)
         P(L)=P(K)
         P(K)=J
         M=2*M
         IF (MOD(BIN,M).GT.0) GO TO 10
  70  CONTINUE
C
      END
```

```
      SUBROUTINE DOUBLE (A,B,C,XA,XB,CE,CF,FA,FB,H,P,ERR,MAXT,MACHP,JF,
     +                   N,NM,NT)
C
C  THIS SUBROUTINE GENERATES AND TESTS PERMUTATIONS FROM DATA FOUND IN
C  EIGENVECTORS CORRESPONDING TO EIGENVALUES OF MULTIPLICITY TWO.
C
C  ON INPUT:
C     A AND B CONTAIN THE ADJACENCY MATRICES OF THE TWO GRAPHS.
C     XA AND XB ARE THE EIGENVECTOR MATRICES OF A AND B.
C     MAXT IS THE MAXIMUM NUMBER OF PERMUTATIONS THAT COULD BE TRIED IF
C        THE OPTIMAL EIGENVECTOR CORRESPONDED TO AN EIGENVALUE OF
C        MULTIPLICTY ONE.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     JF AND JF+1 ARE THE TWO COLUMNS OF XA IN WHICH THE PERTINENT
C        EIGENVECTORS ARE STORED.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF MOST OF THE ARRAYS.
C     NT = 2*NM AND IS THE DIMENSION OF CE AND CF.
C
C  ON RETURN:
C     A, B, XA, AND XB ARE UNCHANGED.
C     P CONTAINS THE LAST PERMUTATION THAT WAS CHECKED AGAINST A AND B.
C        IF ERR=0, THIS PERMUTATION RELATES A AND B.
C     ERR = 4 IF NO PERMUTATION WAS FOUND TO RELATE A AND B BUT NOT ALL
C        POSSIBLE PERMUTATIONS WERE TESTED DUE TO THE LIMIT IMPOSED
C        BY MAXT AND MAXTN.  ERR=0 IF A PERMUTATION SUCCESSFULLY RELATED
C        A AND B.  ERR=-1,1, OR 3 IF ALL POSSIBLE PERMUTATIONS FAILED
C        WHEN TESTED AGAINST A AND B AND THE GRAPHS ARE NOT ISOMORPHIC.
C     MAXT, MACHP, JA, N, NM, AND NT ARE UNCHANGED.
C
C  WORKING VARIABLES:   C,CE,CF,FA,FB,H
C
      INTEGER NM,N,ERR,K,I,J,M,JF,NT,L,MAXT,MAXTN,H(NM),P(NM)
      REAL A(NM,NM),B(NM,NM),C(NM,NM),XA(NM,NM),XB(NM,NM),FA(NM),FB(NM),
     +     CE(NT),CF(NT),MACHP,MACHPN,TM
C
C  RELAX THE REQUIREMENT FOR EQUALITY AS MULTIPLE OPERATIONS ARE TO BE
C  PERFORMED ON THE EIGENVECTOR COMPONENTS.
C
      MACHPN=N*MACHP/5
C
C  SET THE MAXIMUM NUMBER OF TESTS THAT MAY BE PERFORMED ON EACH
C  PROPOSED LINEAR COMBINATION OF THE TWO EIGENVECTORS.
C
      MAXTN=4*MAXT/N
C
      DO 10 J=1,N
         L=J
         IF (ABS(XA(J,JF)).GT.MACHPN.OR.ABS(XA(J,JF+1)).GT.MACHPN)
     +                   GO TO 20
   10 CONTINUE
C
C  COMPUTE ALL POSSIBLE VALUES OF THE TWO COEFFICIENTS AND RECORD THEM
C  IN CE AND CF.
C
   20 TM=XA(L,JF)**2+XA(L,JF+1)**2
      DO 50 J=1,N
         K=2*J
```

```
            CE(K-1)=(XA(L,JF)*XB(J,JF)+XA(L,JF+1)*XB(J,JF+1))/TM
            CF(K-1)=(XA(L,JF+1)*XB(J,JF)-XA(L,JF)*XB(J,JF+1))/TM
            CE(K)=(XA(L,JF)*XB(J,JF)-XA(L,JF+1)*XB(J,JF+1))/TM
            CF(K)=(XA(L,JF+1)*XB(J,JF)+XA(L,JF)*XB(J,JF+1))/TM
C
C  AVOID UNNECESSARY WORK THROUGH DELETION OF DUPLICATED COEFFICIENTS.
C
            IF (J.EQ.1) GO TO 40
               M=K-2
               DO 30 I=1,M
                  IF (ABS(CE(I)-CE(K-1)).LT.MACHP) CE(K-1)=2
                  IF (ABS(CE(I)-CE(K)).LT.MACHP) CE(K)=2
   30          CONTINUE
   40       IF (ABS(CE(K)-CE(K-1)).LT.MACHP) CE(K)=2
C
   50    CONTINUE
C
C  TEST ALL POSSIBLE COEFFICIENTS.
C
         L=0
         DO 80 J=1,NT
C
C  ELIMINATE FROM CONSIDERATION COEFFICIENTS WITH ABSOLUTE VALUE > 1.
C
            IF (ABS(CE(J)).GT.1+MACHPN) GO TO 80
            IF (ABS(CE(J)).GT.1) CE(J)=1
C
C  PERFORM THE LINEAR COMBINATION TO CREATE A PSEUDO EIGENVECTOR.
C
            DO 70 I=1,N
               C(I,JF)=CE(J)*XA(I,JF)+CF(J)*XA(I,JF+1)
   70       CONTINUE
C
C  TREAT THE PSEUDO EIGENVECTOR AS ONE THAT CORRESPONDS TO AN EIGENVALUE
C  OF MULTIPLICITY ONE AND CHECK ALL POSSIBLE PERMUTATIONS.
C
            CALL SINGLE (A,B,C,XB,FA,FB,H,P,ERR,MACHPN,MAXTN,JF,N,NM)
            IF (ERR.EQ.0) RETURN
            IF (ERR.EQ.4) L=4
   80    CONTINUE
         IF (L.EQ.4) ERR=4
C
         RETURN
         END
```

```
      SUBROUTINE SUBVAL (A,B,C,XA,SC,SD,SE,H,P,ERR,MACHP,MAXT,N,NM)
C
C  THIS SUBROUTINE GENERATES AND COMPARES THE SPECTRA OF THE ORDER N-1
C  SUBGRAPHS OF THE TWO ORIGINAL GRAPHS.  AN ATTEMPT IS MADE TO GENERATE
C  A PERMUTATION BETWEEN THE TWO GRAPHS BASED ON THIS DATA.
C
C  ON INPUT:
C     A AND B CONTAIN THE ADJACENCY MATRICES OF THE TWO GRAPHS.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     MAXT IS THE MAXIMUM NUMBER OF PERMUTATIONS THAT COULD BE TRIED IF
C        THE OPTIMAL EIGENVECTOR CORRESPONDED TO AN EIGENVALUE OF
C        MULTIPLICTY ONE.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A AND B ARE UNCHANGED.
C     C CONTAINS SPECTRA DUPLICATION INFORMATION.  THE I TH ROW OF C
C        CONTAINS THE INDECIES OF ALL NODES IN A WHICH WHEN UNIQUELY
C        MARKED PRODUCE SPECTRA IDENTICAL TO THAT PRODUCED WHEN THE I TH
C        NODE OF A WAS UNIQUELY MARKED.
C     XA CONTAINS THE N SPECTRA GENERATED FROM THE SUBGRAPHS OF A.  EACH
C        ROW CONTAINS ONE SPECTRA.
C     SC OF I IS THE NUMBER OF NODES IN A, INCLUDING I, THAT PRODUCED
C        SPECTRA IDENTICAL TO THAT OF I.
C     P CONTAINS A PERMUTATION OF THE SET OF SPECTRA OF SUBGRAPHS OF A
C        INTO THOSE OF B.  IF ERR=N+1, THIS IS A PERMUTATION OF A
C        INTO B.
C     ERR IS NEGATIVE IF AN ERROR OCCURS IN THE EISPACK SOFTWARE.  IF
C        THE SETS OF SPECTRA ARE NOT ISOMORPHIC, ERR CONTAINS THE INDEX
C        OF THE NODE IN B WHICH WHEN UNIQUELY MARKED, PRODUCED A SPECTRA
C        THAT COULD NOT BE MATCHED WITH A SPECTRA FROM THE SUBGRAPHS OF
C        A.  ERR=0 IF THE SETS OF SPECTRA WERE ISOMORPHIC BUT THREE OR
C        MORE NODES OF A, WHICH WHEN UNIQUELY MARKED, PRODUCED IDENTICAL
C        SPECTRA.  ERR=N+1 IF A PERMUTATION RELATING A AND B WAS FOUND.
C        ERR=N+4 IF ALL POSSIBLE PERMUTATIONS FAILED TO RELATE A AND B.
C        ERR=N+5 IF MAXT PERMUTATIONS WERE GENERATED AND TESTED WITHOUT
C        SUCCESS.
C     MACHP, MAXT, N, AND NM ARE UNCHANGED.
C
C  WORKING VARIABLES:    SD,SE,H
C
      INTEGER NM,N,I,J,K,L,ERR,M,MAXT,LT,H(NM),P(NM)
      REAL A(NM,NM),B(NM,NM),C(NM,NM),XA(NM,NM),SC(NM),SD(NM),SE(NM),FN,
     +     MACHP
C
C  FN IS THE FUNCTION FOR X-1 IN THE 1,X FORM OF THE ADJACENCY MATRIX.
C
      FN=N/2
C
C  GENERATE THE SPECTRA OF ALL ORDER N-1 SUBGRAPHS OF A.
C
      DO 50 I=1,N
         P(I)=0
C
C  COPY A INTO C.
C
         DO 20 K=1,N
```

```
            DO 10 J=1,N
               C(K,J)=A(K,J)
 10         CONTINUE
 20      CONTINUE
C
C  UNIQUELY MARK THE I TH NODE OF C=A AND ALL NODES CONNECTED TO IT.
C
         DO 30 J=1,N
            C(J,J)=C(J,J)+C(I,J)
            C(I,J)=3*FN
            C(J,I)=3*FN
 30      CONTINUE
         C(I,I)=C(I,I)+FN
C
C  COMPUTE THE SPECTRA  -  C IS DESTROYED IN THIS COMPUTATION.
C
         CALL TRED1 (NM,N,C,SC,SD,SE)
         CALL TQL1 (N,SC,SD,ERR)
         ERR=-ERR
         IF (ERR.NE.0) RETURN
         DO 40 J=1,N
            XA(I,J)=SC(J)
 40      CONTINUE
 50   CONTINUE
C
C  SEQUENTIALLY COMPUTE THE SPECTRA OF THE N-1 ORDER SUBGRAPHS OF B,
C  CHECKING THAT EACH CAN BE MATCHED WITH A SIMILAR SPECTRA FROM THE
C  SUBGRAPHS OF A.
C
      DO 110 I=1,N
         DO 70 K=1,N
            DO 60 J=1,N
               C(K,J)=B(K,J)
 60         CONTINUE
 70      CONTINUE
         DO 80 J=1,N
            C(J,J)=C(J,J)+C(I,J)
            C(I,J)=3*FN
            C(J,I)=3*FN
 80      CONTINUE
         C(I,I)=C(I,I)+FN
         CALL TRED1 (NM,N,C,SC,SD,SE)
         CALL TQL1 (N,SC,SD,ERR)
         ERR=-ERR
         IF (ERR.NE.0) RETURN
         DO 100 J=1,N
            IF (P(J).GT.0) GO TO 100
               DO 90 K=1,N
                  IF (ABS(XA(J,K)-SC(K)).GT.MACHP*(1+ABS(SC(K))))
     +                        GO TO 100
 90               CONTINUE
                  P(J)=I
                  GO TO 110
100      CONTINUE
         ERR=I
         RETURN
110   CONTINUE
C
```

```
      LT=0
      L=0
C
C  COMPUTE AND RECORD THE DATA PERTAINING TO SPECTRA DUPLICATION IN A.
C
      DO 140 I=1,N
         M=0
         DO 130 J=1,N
            DO 120 K=1,N
               IF (ABS(XA(I,K)-XA(J,K)).GT.MACHP*(1+ABS(XA(I,K))))
     +                GO TO 130
 120        CONTINUE
               M=M+1
               C(I,M)=P(J)
               IF (M.EQ.3) L=1
               IF (L.EQ.1.OR.I.GE.J) GO TO 130
C
C  RECORD DUPLICATED PAIRS OF SPECTRA WITH THE HOPE THAT THERE ARE NO
C  TRIPLICATES AND SUBROUTINE TEST CAN BE USED TO FIND A PERMUTATION OF
C  A INTO B OR PROVE THAT ONE DOES NOT EXIST.
C
                  LT=LT+1
                  H(2*LT-1)=I
                  H(2*LT)=J
 130     CONTINUE
         SC(I)=M
 140  CONTINUE
C
      IF (L.EQ.1) RETURN
      CALL TEST (A,B,H,P,ERR,MAXT,LT,N,NM)
      ERR=ERR+N+1
C
      RETURN
      END
```

```
      SUBROUTINE SUBVEC (A,B,C,XA,XB,DA,DB,EA,EB,FA,FB,SC,SD,SE,G,H,P,
     +                   ERR,MACHP,MAXT,N,NM)
C
C  THIS SUBROUTINE ATTEMPTS TO GENERATE A PERMUTATION BETWEEN A AND B
C  THROUGH EXAMINATION OF EIGENVECTORS CORRESPONDING TO EIGENVALUES OF
C  MULTIPLICITY ONE OF ORDER N-1 SUBGRAPHS OF A AND B.
C
C  ON INPUT:
C     A AND B CONTAIN THE ADJACENCY MATRICES OF THE TWO GRAPHS.
C     C AND SC CONTAIN SPECTRA DUPLICATION INFORMATION.
C     P CONTAINS A PERMUTATION OF THE SET OF ORDER N-1 SPECTRA OF A INTO
C        THOSE OF B.
C     MACHP IS THE PRECISION WITHIN WHICH EQUALITY IS TO BE TESTED.
C     MAXT IS THE MAXIMUM NUMBER OF PERMUTATIONS THAT COULD BE TRIED IF
C        THE OPTIMAL EIGENVECTOR CORRESPONDED TO AN EIGENVALUE OF
C        MULTIPLICTY ONE.
C     N IS THE ORDER OF THE GRAPHS.
C     NM IS THE DIMENSION OF THE ARRAYS.
C
C  ON RETURN:
C     A AND B ARE UNCHANGED.
C     C AND SC ARE DESTROYED.
C     P CONTAINS A PERMUTATION OF THE COMPONENTS OF AN EIGENVECTOR OF
C        A SUBGRAPH OF A INTO AN EIGENVECTOR OF A SUBGRAPH OF B.  IF
C        ERR=0 THEN THE PERMUTATION RELATES A AND B.
C     ERR = 0 IF A PERMUTATION OF A INTO B WAS FOUND.  ERR=1 IF ALL
C        POSSIBLE PERMUTATIONS FAILED TO RELATE A AND B.  ERR=2 IF ALL
C        PERMUTATIONS CHECKED FAILED BUT NOT ALL POSSIBLE PERMUTATIONS
C        WERE TESTED DUE TO THE RESTRICTIONS IMPOSED BY MAXT.  ERR=3 IF
C        ALL EIGENVECTORS CORRESPONDING TO EIGENVALUES OF MULTIPLICITY
C        ONE HAD THREE OR MORE IDENTICAL COMPONENTS.  ERR IS NEGATIVE IF
C        AN ERROR CONDITION WAS RAISED IN THE EISPACK SOFTWARE.
C     MACHP, MAXT, N, AND NM ARE UNCHANGED.
C
C  WORKING VARIABLES:   XA,XB,DA,DB,EA,EB,FA,FB,SD,SE,G,H
C
      INTEGER NM,N,MAXT,ERR,I,J,K,L,M,LT,LM,PN,JA,G(NM),H(NM),P(NM),NN
      REAL A(NM,NM),B(NM,NM),C(NM,NM),XA(NM,NM),XB(NM,NM),DA(NM),DB(NM),
     +     EA(NM),EB(NM),FA(NM),FB(NM),SC(NM),SD(NM),SE(NM),MACHP,MACHPD
     +     ,EIGVAL,LB,UB,RD,FN
C
C  FN IS THE FUNCTION FOR X-1 IN THE 1,X FORM OF THE ADJACENCY MATRIX.
C
      FN=N/2
      NN=N-1
      PN=1000
C
C  SEARCH FOR AN OPTIMAL EIGENVECTOR AMONG THOSE THAT CORRESPOND TO AN
C  EIGENVALUE OF MULTIPLICITY ONE.
C
      DO 90 I=1,N
C
C  UNIQUELY MARK THE I TH NODE OF A AND THE NODES CONNECTED TO IT.
C
         DO 10 J=1,N
            DA(J)=A(I,J)
            A(J,J)=A(J,J)+DA(J)
            A(I,J)=3*FN
```

```
            A(J,I)=3*FN
   10       CONTINUE
            A(I,I)=A(I,I)+FN
C
C  COMPUTE THE EIGENVALUES AND EIGENVECTORS OF THIS SUBGRAPH.  A IS NOT
C  DESTROYED.
C
            CALL TRED2 (NM,N,A,DB,FB,XA)
            CALL TQL2 (NM,N,DB,FB,XA,ERR)
            ERR=-ERR
            IF (ERR.NE.0) RETURN
C
C  RESTORE A TO ITS ORIGINAL VALUE.
C
            DO 20 J=1,N
               A(J,J)=A(J,J)-DA(J)
               A(I,J)=DA(J)
               A(J,I)=DA(J)
   20       CONTINUE
C
C  SEARCH FOR THE OPTIMAL EIGENVECTOR OF THOSE JUST COMPUTED.
C
            DO 80 J=1,N
               LT=SC(I)
               MACHPD=MACHP*(1+ABS(DB(J)))
C
C  CHECK THAT THE ASSOCIATED EIGENVALUE HAS MULTIPLICITY ONE.
C
               IF (J.EQ.1) GO TO 30
                  IF (ABS(DB(J)-DB(J-1)).LT.MACHPD) GO TO 80
                     IF (J.EQ.N) GO TO 40
   30          IF (ABS(DB(J)-DB(J+1)).LT.MACHPD) GO TO 80
C
C  FIND AND RECORD THE NUMBER OF DUPLICATED PAIRS OF COMPONENTS.  IF A
C  TRIPLICATE IS FOUND, DISCONTINUE CONSIDERATION OF THAT EIGENVECTOR.
C
   40             DO 60 L=1,NN
                     M=L+1
                     LM=0
                     DO 50 K=M,N
                        IF (ABS(XA(L,J)-XA(K,J)).GT.MACHP) GO TO 50
                           IF (LM.EQ.1) GO TO 80
                              LM=1
                              LT=LT+1
   50                CONTINUE
   60             CONTINUE
                  IF (LT.GE.PN) GO TO 80
C
C  UPDATE CURRENT OPTIMAL EIGENVECTOR INFORMATION.
C
                     PN=LT
                     JA=I
                     EIGVAL=DB(J)
                     DO 70 K=1,N
                        SD(K)=XA(K,J)
   70                CONTINUE
C
C  IF THE EIGENVECTOR IS SUITABLE  -NOT NECESSARILY OPTIMAL-  TERMINATE
```

```
C  THE SEARCH.
C
                    IF (LT.LT.10) GO TO 100
  80      CONTINUE
  90    CONTINUE
C
C  IF NO SUITABLE EIGENVECTOR COULD BE FOUND, TERMINATE THIS PORTION
C  OF THE ALGORITHM WITH APPROPRIATE MESSAGE.
C
       IF (PN.LT.1000) GO TO 100
          ERR=3
          RETURN
 100   MACHPD=MACHP*(1+ABS(EIGVAL))
       LB=EIGVAL-MACHPD
       UB=EIGVAL+MACHPD
C
C  STORE THE SELECTED EIGENVECTOR IN THE FIRST COLUMN OF XA.
C
       DO 110 I=1,N
          XA(I, 1)=SD(I)
 110   CONTINUE
C
C  SELECTIVELY STORE THE PERTINENT INFORMATION HELD IN C AND SC IN ORDER
C  TO FREE STORAGE FOR FUTURE USE.
C
       M=SC(JA)
       DO 120 I=1,M
          G(I)=C(JA,I)
 120   CONTINUE
C
C  FIND AND EXAMINE THE CORRESPONDING EIGENVECTOR IN THE SUBGRAPHS OF B
C  WHICH HAVE SPECTRA MATCHING THE SPECTRA GENERATED FROM THE SUBGRAPH
C  OF A THAT PRODUCED THE OPTIMAL EIGENVECTOR.
C
       DO 170 L=1,M
C
C  COPY B INTO C.
C
          DO 140 I=1,N
             DO 130 J=1,N
                C(I,J)=B(I,J)
 130         CONTINUE
 140      CONTINUE
C
C  FIND AND UNIQUELY MARK THE APPROPRIATE NODE IN B.
C
          I=G(L)
          DO 150 J=1,N
             C(J,J)=C(J,J)+C(I,J)
             C(I,J)=3*FN
             C(J,I)=3*FN
 150      CONTINUE
          C(I,I)=C(I,I)+FN
C
C  FIND THE DESIRED EIGENVECTOR  -  C IS DESTROYED.
C
          CALL TRED1 (NM,N,C,DA,DB,SC)
          CALL BISECT (N,0,DA,DB,SC,LB,UB,1,PN,RD,LM,ERR,SD,SE)
```

```
         ERR=-2*ERR-PN+1
         IF (ERR.NE.0) RETURN
         CALL TINVIT (NM,N,DA,DB,SC,PN,RD,LM,FB,ERR,SD,SE,EA,EB,FA)
         ERR=-ERR
         IF (ERR.NE.0) RETURN
         CALL TRBAK1 (NM,N,C,DB,PN,FB)
C
C  STORE THE EIGENVECTOR IN THE FIRST COLUMN OF XB.
C
         DO 160 J=1,N
            XB(J, 1)=FB(J)
 160     CONTINUE
C
C  SINCE BOTH EIGENVECTORS ARE STORED IN COLUMN ONE OF MATRIX ARRAYS,
C  SUBROUTINE SINGLE CAN BE USED TO GENERATE AND TEST UP TO MAXT
C  PERMUTATIONS.
C
         CALL SINGLE (A,B,XA,XB,DA,DB,H,P,ERR,MACHP,MAXT, 1,N,NM)
         IF (ERR.EQ.0) RETURN
         IF (ERR.EQ.4) NN=N
 170  CONTINUE
      ERR=1
      IF (NN.EQ.N) ERR=2
C
      RETURN
      END
```

4. Computational Experience

The UNIVAC 1108 computer at the National Bureau of Standards was used to test the computer program on well over 200 pairs of graphs specifically selected for use because of their difficulty. The results have been encouraging. A Brookhaven National Laboratories group has compiled [2] a list of 149 pairs of nonisomorphic graphs of order varying from 7 to 10 that have 0,1 adjacency matrices with identical spectra. These cospectral graphs arise in connection with the energy field of a Heisenberg model ferromagnet. The computer program previously listed distinguished each of the 149 pairs in a total time of 3.9 cpu seconds or, equivalently, each pair of graphs was determined to be nonisomorphic in approximately 0.026 cpu seconds. Of the graphs, 109 failed the valence test, 28 the node duplication test, and the remaining 12 pairs failed to have $1,x$ modified adjacency matrices with identical spectra.

With some modification the algorithm can be programmed to sort a large set of graphs into subsets characterized by node valences, node duplication, $1,x$ modified adjacency matrix spectra, and in more difficult cases, submatrix spectra. Once this is accomplished, the groups in each subset can be pairwise checked for isomorphism using the program listed in this paper. This procedure was employed on a collection of 35 graphs ranging in order from 5 to 9. Many of the graphs selected were known to be isomorphic and all had large automorphism groups, thus making it as difficult as possible for the algorithm to find permutations or to determine that none exist. By means of a simplified form of the sorting procedure, the 35 graphs were divided into 19 subsets in approximately 5 cpu seconds. In this particular case, the spectra test was a sufficient condition for graph isomorphism. To be sure, however, permutations were generated between each pair of graphs in each subset. A total of 16 permutations were generated, 13 by means of SUBROUTINE SINGLE and 3 requiring SUBROUTINE DOUBLE, in a total time of 1.6 cpu seconds.

None of the graphs yet tested, however, required the use of SUBROUTINE SUBVAL or SUBROUTINE SUBVEC. In response to a July, 1975 query about the existence of more challenging graphs, D. Corneil provided 16 pairwise nonisomorphic, order 25 graphs, any two of which required approximately 60 seconds on an IBM 370 (no model number was mentioned) to be distinguished by then current methods. The graphs were 3-strongly regular and stochastic with node valence 12. None of the graphs has duplicate nodes and each had identical $1,x$ adjacency matrix spectra. The submatrix spectra test, however, successfully sorted the group into 8 subsets, each containing two graphs. It is interesting to note that the two graphs contained in each subset were duals of each other. This initial sorting was accomplished in just 86 cpu seconds. When the algorithm was run on the 8 pairs of graphs, 5 were separated by means of the submatrix eigenvector test. The test was inconclusive on the remaining 3 pairs of graphs. The total run time was 150 cpu seconds. Of this total, 86 cpu seconds was spent distinguishing the 5 pairs and 64 cpu seconds on the unsuccessful effort to resolve the final 3 pairs. In addition, the algorithm was applied to 5 of the graphs and constructed permutations. The program produced a permutation relating each of the 5 pairs of graphs by means of SUBROUTINE SUBVEC in 81 cpu seconds.

Approximately 90 percent of the time required by the program is used in the calculation of eigenvalues and eigenvectors. Thus selection of optimal eigenvalue and eigenvector computing software is critical when time restrictions are sizeable. Since the EISPACK software referenced in the listing of the algorithm proved to be satisfactory for our needs, no significant effort was made to find more efficient software.

Tests on large numbers of random graphs were not conducted as it was felt that such tests would yield little meaningful data. Experience gained from working with the eigenvectors and eigenvalues of many graphs showed that only a relatively minute number of pairs of graphs exist for which the question of isomorphism is at all challenging for the algorithm and no such graphs are likely to occur in even a very large sample of randomly generated graphs. For the average pair of isomorphic graphs of order n, it is estimated that a permutation will be generated in well under $10^{-4}n^3$ cpu seconds UNIVAC 1108 time. Even faster times are achievable for random nonisomorphic pairs of graphs or through modifications in programming which first sort graphs by node valences, node duplicate structure $1,x$ modified adjacency matrix spectra, and in special cases (such as Corneil's collection of order 25 graphs) submatrix spectra, and then use the algorithm described in this paper to further reduce the subsets into classes for which the graphs are pairwise isomorphic.

Thus, with efficient application of the algorithm it is possible to divide a very large number of graphs into isomorphism equivalence classes in a reasonable amount of time. It is further possible to test pairs of very large graphs for isomorphism and if one exists, to generate permutations between two very large graphs.

5. References

[1.] Baker, G., Drum shapes and isospectral graphs, J. Math. Physics **7,** 2238–2242 (1966).

[2.] Baker, G., et al., A Data Compendium of Linear Graphs with Application to the Heisenberg Model, Brookhaven National Laboratory Report, 1967.

[3.] Corneil, D., and Gotlieb, C., An Efficient algorithm for graph isomorphism, JACM **17,** 51–64 (1970).

[4.] Corneil, D., The Analysis of Graph Theoretical Algorithms, Proc. Fifth S-E Conference on Combinatorics, Graph Theory and Computing, pp. 3–38 (1974).

[5.] Harary, F., King, C., Mowshowitz, A., and Read, R., Cospectral graphs and digraphs, Bull. London Math. Soc. **3,** 321–328 (1971).

[6.] Johnson, C., and Newman, M., A Note on Cospectral Graphs, submitted.

[7.] Mathon, R., and Corneil, D., Private Communications 1975–76.

[8.] Schmidt, D., and Druffel, L., A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices, JACM **23,** 433–445 (1976).

[9.] Smith, B., et al., Matrix Eigensystem Routines — EISPACK Guide, Lecture Notes in Computer Science 6, (Springer-Verlag, New York, 1974).

(Paper 80B4–457)