Computer Generation of Automorphism Groups of Weighted Graphs

K. Balasubramanian

Department of Chemistry and Biochemistry, Arizona State University, Tempe, Arizona 85287-1604

Received April 7, 1994®

Computational techniques are described for the automorphism groups of edge-weighted graphs. Fortran codes based on the manipulation of weighted adjacency matrices are used to compute the automorphism groups of several edge-weighted graphs. The code developed here took $37^1/_2$ min of CPU time to generate 1 036 800 permutations in the automorphism group of an edge-weighted graph.

I. INTRODUCTION

Computational methods of the symmetry (automorphism) groups of graphs have been the topics of several studies.[1-20] The computer generation of the automorphism groups of graphs is a very important problem since most of the computer techniques for structure enumeration and spectral simulation require the automorphism groups of the associated graphs. For example, in the computer-assisted structure elucidation methods, two-dimensional connection tables are obtained. Any further chemical application or data reduction often requires the knowledge of the automorphism group. To illustrate, the number of ^{13}C-NMR signals of the molecules associated with the connectivity table depends on the vertex equivalencies which in turn depend on the automorphisms of the graph. The enumeration of the stereoisomers from the structural isomer information requires the permutations in the automorphism group of the graph. Spectral simulation methods also need the automorphism group. The enumeration of ESR hyperfine patterns can be more readily accomplished if the permutations in the automorphism group are determined.

One should recognize that the symmetry group of a molecule is very differnet from the automorphism group of a graph. The symmetry group of the point group of a molecule comprises proper (C_n) and improper rotations (S_n) of the molecule which leave the structure invariant, the center of inversion, and planes of symmetry being special cases of S_n. The automorphism group of a graph, on the other hand, comprises permutations of the vertices of the graph such that neither existing connections between the vertices are broken nor new connections (which are not in the graph) are introduced by the permutations. There are no three-dimensional perceptions or implications in characterizing the automorphism groups. Although the molecular symmetry group can be cast into a group of permutation–inversion (PI) operations, the PI group of the molecule, in general, is not isomorphic to the automorphism group of the associated molecular graph.

The computer generation of automorphism groups of graphs is a computationally intensive problem. As noted in recent studies, there is no unique efficent technique for all graphs. In fact, an efficient method for one graph performs poorly for another graph. The most general procedure in the worst case has to go through $n!$ computations, where n is the number of vertices. Several studies have addressed computational procedures for the automorphism groups of graphs.[1-3,11,17-20]

An ordinary graph carries no distinction among bonds or among vertices. That is, all vertices carry the same weights. Likewise, all edges carry the same weights. However, for real-world chemical and spectroscopic applications one needs to utilize graphs which contain different weights for different edges, for example, to discriminate single and double bonds. Likewise, chemically different atoms needs to be distinguished by labeling the vertices of graphs.

In the context of NMR and multiple NMR spectroscopy, the nuclear spin couplings can be represented by graphs called the NMR graphs.[2,3] In these examples, the vertices are the various nuclei in the molecule, while the edges are either isotopic J-coupling constants or dipolar coupling constants. The nuclear coupling constants between the various nuclei in a molecule are often different as they depend on the Euclidean distances between the nuclei. Consequently, it is necessary to label the edges of the graphs with the appropriate nuclear spin–spin coupling constants. The same is true for the hyperfine structure in the ESR spectra. The automorphism groups of ordinary graphs are often different from the edge (vertex)-weighted graphs.

The objective of this investigation is to focus on computational techniques and codes for the computer generation of the automorphism groups of edge-weighted graphs. We consider several edge-weighted graphs. It is shown that the code outlined here is efficient for these graphs. Several examples of edge-weighted graphs are shown for which the edge-weighted automorphism groups differ from the ordinary automorphism groups.

II. AUTOMORPHISM GROUPS OF EDGE-WEIGHTED GRAPHS

The adjacency matrix of an edge-weighted graph is defined as

$$A_{ij} = \begin{cases} w_{ij} & \text{if the vertices } i \text{ and } j \text{ are connected with the edge weight } w_{ij} \\ 0 & \text{otherwise} \end{cases}$$

This differs from the ordinary adjacency matrix in that all weights of the ordinary adjacency matrix are unities for an ordinary graph. This important difference in the edge weights introduces a contrast in the automorphisms of an edge-weighted graph compared to an ordinary graph. The automorphism group consists of permutations of the vertices which leave the graph "invariant". For an edge-weighted graph this can be generalized using the following precise mathematical definition.

The automorphism group of an edge-weighted graph consists of permutations of the vertices of the graph whose permutation matrices $\mathbf{P}$ satisfy

$$\mathbf{A} = \mathbf{P}^T\mathbf{A}\mathbf{P}$$

® Abstract published in *Advance ACS Abstracts*, June 15, 1994.

0095-2338/94/1634-1146$04.50/0

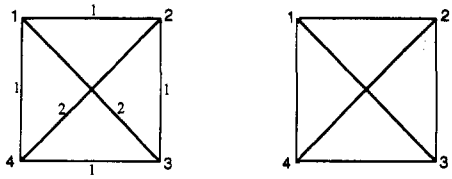

Figure 1. Two graphs on four vertices, one edge-weighted (a) and the other without edge weights (b).

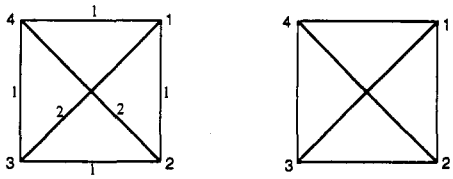

Figure 2. Effect of vertex permutations (1 2 3 4) on both graphs. Note that the permutation (1 2 3 4) belongs to the automorphism groups of both graphs.

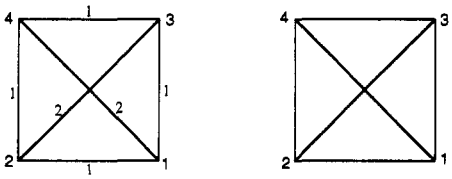

Figure 3. Effect of permutations (1 3 2 4) on both graphs. The permutation (1 3 2 4) does not belong to the automorphism group of graph a, while it does belong to the automorphism group of graph b.

where **A** is the adjacency matrix of the edge-weighted graph under consideration, **P** is an $n \times n$ permutation matrix composed of 0's and 1's, and $\mathbf{P}^T$ is the transpose of the matrix **P**.

We now illustrate the automorphism group of an edge-weighted graph and contrast it with the automorphism group of an ordinary graph. Consider the complete edge-weighted graph K_4 shown in Figure 1a. The corresponding ordinary complete graph without edge weights is shown in Figure 1b. The numbers shown on the edges are the edge weights for the graph in Figure 1a.

We consider the effect of permutation (1 2 3 4) which is also represented in the two-row notation as follows

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$$

Figure 2a (Figure 2b) shows the graph obtained by permuting the vertices of Figure 1a (Figure 1b) by the above permutation. If we compare Figure 2a with Figure 1a, we find that the edge-weighted graph in Figure 2a is isomorphic or equivalent to Figure 1a. That is, although the vertices themselves have changed their positions, the edge weights between various vertices in both Figure 1a and Figure 2a are the same. There are six edges in the graphs in Figure 1a and Figure 2a. The weights between vertices 1 and 2, 1 and 3, 1 and 4, 2 and 3, 2 and 4, and 3 and 4 are the same in both Figure 1a and Figure 2a. Likewise Figure 1b and Figure 2b are equivalent in that no bond is made or broken. Consequently, the permutation (1 2 3 4) is in the automorphism group of both the ordinary graph and the edge-weighted graph.

The effect of the above permutation on the edges can be seen in terms of the matrices as follows. The adjacency matrices are shown as follows for Figure 1a,b.

$$\begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix} \quad \text{for Figure 1a} \qquad \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \quad \text{for Figure 1b}$$

The permutation (1 2 3 4) has the following permutation matrices.

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{P}^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Note that in the ith row of the permutation matrix, the jth column is 1 if the permutation carries i to j; all other entreis in that row are zeros. We have shown as follows, the effect of multiplying **P** and $\mathbf{P}^T$ on the two adjacency matrices.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix} \quad \text{(Figure 1a)}$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \quad \text{(Figure 1b)}$$

As seen from the above matrix multiplications the adjacency matrices are the same for Figure 1a (Figure 1b) before and after the matrix multiplications. Consequently, the permutation (1 2 3 4) is a member of the automorphism groups of both graphs in Figure 1a,b.

We consider the permutation (1 3 2 4) which is represented in the two-row notation as

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$$

The effects of this permutation on the graphs in Figure 1a,b are shown in Figure 3a,b, respectively.

The permutation matrix and its transpose for this permutation are as follows.

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{P}^T = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

We multiply both **P** and $\mathbf{P}^T$ with the adjacency matrixes for Figure 1a,b

$$\begin{bmatrix} 0&0&0&1\\0&0&1&0\\1&0&0&0\\0&1&0&0 \end{bmatrix}\begin{bmatrix} 0&1&2&1\\1&0&1&2\\2&1&0&1\\1&2&1&0 \end{bmatrix}\begin{bmatrix} 0&0&1&0\\0&0&0&1\\0&1&0&0\\1&0&0&0 \end{bmatrix}$$

$$=\begin{bmatrix} 0&0&0&1\\0&0&1&0\\1&0&0&0\\0&1&0&0 \end{bmatrix}\begin{bmatrix} 1&2&0&1\\2&1&1&0\\1&0&2&1\\0&1&1&2 \end{bmatrix}$$

$$=\begin{bmatrix} 0&1&1&2\\1&0&2&1\\1&2&0&1\\2&1&1&0 \end{bmatrix} \quad \text{(Figure 1a)}$$

Note that in this case the matrix obtained after matrix multiplications is not equal to the matrix before multiplication. Graphically the weighted graph obtained by the permutation of the vertices according to this permutation (Figure 3a) is not equivalent to the original graph (Figure 1a). For example, the edge weight between vertices 1 and 4 is 2 in Figure 3a, while in the unpermuted graph (Figure 1a), the corresponding weight is 1. Likewise the weight between vertices 2 and 3 is 2 in the permuted graph, while the corresponding weight is 1 in the unpermuted graph (Figure 1a). Hence the permutation (1 3 2 4) does not belong to the automorphism group of the weighted graph.

We apply the same permutation to the unweighted complete graph (Figure 1b). Diagramatically, the effect of the vertex permutation (1 3 2 4) is shown in Figure 3b. The matrix multiplication is shown as follows.

$$\begin{bmatrix} 0&0&0&1\\0&0&1&0\\1&0&0&0\\0&1&0&0 \end{bmatrix}\begin{bmatrix} 0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0 \end{bmatrix}\begin{bmatrix} 0&0&1&0\\0&0&0&1\\0&1&0&0\\1&0&0&0 \end{bmatrix}$$

$$=\begin{bmatrix} 0&0&0&1\\0&0&1&0\\1&0&0&0\\0&1&0&0 \end{bmatrix}\begin{bmatrix} 1&1&0&1\\1&1&1&0\\1&0&1&1\\0&1&1&1 \end{bmatrix}$$

$$=\begin{bmatrix} 0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0 \end{bmatrix} \quad \text{(Figure 1b)}$$

The matrix obtained after the permutation matrices multiplication is equal to the original adjacency matrix. Hence the permutation (1 3 2 4) belongs to the automorphism group of the unweighted complete graph (Figure 1b), while the same permutation does not belong to the automorphism group of the edge-weighted graph in Figure 1a. This simple example illustrates the difference between the normal automorphism permutation and the automorphism permutation of a weighted graph.

The set of all automorphism permutations forms a group for both unweighted and weighted graphs. The automorphism group for the graph in Figure 1a is isomorphic with the dihedral group D_4 consisting of eight permutations:

{(1)(2)(3)(4), (1 2 3 4), (1 3)(2 4), (1 4 3 2),
(1)(3)(2 4), (1 3)(2)(4), (1 2)(3 4), (1 4)(2 3)}

The automorphism group of the graph in Figure 1b, on the other hand, is the S_4 permutation group comprising all 24 permutations of 4 objects. This set consists of all permutations in D_4 enumerated above and the following 16 permutations:

(1 3 2 4), (1 4 2 3), (1 2 4 3), (1 3 4 2), (1)(2 3 4),
(1)(2 4 3), (2)(1 3 4), (2)(1 4 3), (3)(1 2 4), (3)(1 4 2),
(4)(1 2 3), (4)(1 3 2), (1)(2)(3 4), (1)(4)(2 3),
(2)(3)(1 4), and (3)(4)(1 2)

Evidently the automorphism group of an edge-weighted graph would be a subgroup of the automorphism group of the corresponding unweighted graph. Whenever all edges are weighted with the same weight, the automorphism group of the edge-weighted graph becomes the automorphism group of the unweighted graph.

III. COMPUTATIONAL TECHNIQUES

In an earlier investigation the current author[1] recognized the importance of using array index manipulation instead of matrix multiplication for the generation of the automorphism groups. This is, the matrix mutliplication technique illustrated above, although useful in defining the automorphism permutation, is not the best suited for computational implementation. This is because multiplication of two $n \times n$ matrices requires n^3 multiplication operations. Suppose the new matrix **B** is obtained as follows:

$$\mathbf{B} = \mathbf{P}^T\mathbf{A}\mathbf{P}$$

where **P** is the permutation matrix which corresponds to the permutation p of the following form:

$$\begin{bmatrix} 1 & 2 & 3 & \cdots & i & \cdots & n \\ p_1 & p_2 & p_3 & \cdots & p_i & \cdots & p_n \end{bmatrix}$$

It can be shown that the ijth matrix element of **B** is related to the original adjacency matrix **A** of the edge-weighted graph as follows:

$$B_{ij} = A_{p_i p_j}$$

Therefore all the matrix elements of the B matrix can be found in the **A** matrix itself without performing matrix multiplications. This is the method we use in the current code for the automorphism group of the edge-weighted graph. If

$$A_{p_i p_j} = A_{ij} \qquad \text{for all } i = 1, n$$

then the permutation p is included in the automorphism group of the edge-weighted graph under consideration; otherwise it is disregarded.

The code also uses the automorphism equivalence classes of edge-weighted graphs. Two vertices of an edge-weighted graph are considered to be equivalent if they belong to the same orbit in any one of the permutations of the automorphism group of the edge-weighted graph. There are several techniques described in the literature for the automorphism partitioning of vertices of ordinary graphs into equivalence classes.[3-10] These techniques do not require the automorphism permutations. They use methods such as the principal eigenvector analysis, the Morgan algorithm, the canonical labeling method, etc. It would be interesting to see how these technqiues perform for edge-weighted graphs. Such investigations can be topics of future studies. Our code accepts equivalence class information as input.

Suppose the edge-weighted graph under investigation has n_1 vertices in the first equivalence class, n_2 vertices in the second equivalence class, n_m vertices in the mth equivalence class, etc. The automorphism group, Γ, of the edge-weighted graph is then a subgroup of $S_{n_1} \times S_{n_2} \times \ldots \times S_{n_m}$, where S_{n_i}

is a permutation group containing n_i elements. Symbolically,

$$\Gamma \subseteq S_{n_1} \times S_{n_2} \times \ldots \times S_{n_m}$$

$$\sum_{i=1}^{m} n_i = n$$

Our code takes full advantage of the above fact so that the number of searches can be minimized. The code generates all $n_1!n_2!\ldots n_m!$ permutations of n vertices, and the condition that $A_{p_i p_j} = A_{ij}$ is checked for each of the permutations. Note that no matrix multiplications are done.

As described before, the properties of a group such as closure and inverse can be utilized to accelerate the performance of the code. The code execution can be stopped after a specified CPU time is reached. The permutations obtained up to that time can be extended using closure, generator, and inverse properties. The code can be restarted using the farthest permutation generated from the identity permutation in the permutation generation tree. This calls for evaluation of the genealogical relationship of the various permutations in the permutation generator tree and choosing the farthest one as the current permutation to restart the automorphism group generation.

IV. COMPUTATIONAL IMPLEMENTATIONS

In this section we choose a set of graphs of varied complexities to apply our code for the generation of automorphism groups. In some cases we also compare the automorphism groups of the corresponding unweighted graphs. In each case we also give the CPU time. We show eight edge-weighted graphs labeled I–VIII in Figure 4 for computational consideration.

Table 1 shows the number of permutations in the automorphism groups of the edge-weighted graphs in Figure 4 as generated by our computer together with their CPU times. As seen from Table 1, four of the eight graphs in Figure 4 took less than a second of CPU time on an IBM RS 6000-580 workstation. Some of the graphs shown in Figure 4 were considered to be "pathological" in the literature.[3,15] It is noteworthy that our code had no difficulty in generating the correct automorphism groups for these graphs.

The labeling of the edges of the cube graph I in Figure 4 reduces the automorphism group from a group of order 48 for an unweighted cube graph to a group containing 16 permutations. Likewise consider the Petersen graph (graph V in Figure 4). The automorphism group of the ordinary unweighted Petersen graph is the S_5 group which consists of 120 permutations. This reduces to a group containing 20 permutations when the edges are weighted according to graph V, Figure 4.

Graph IV in Figure 4 appears in NMR spectroscopy.[23] It is the NMR graph of the six protons of ethane. The protons on the same carbon atom have one set of coupling constants, while the protons attached to different carbon atoms have another set of coupling constants. Consequently, we weigh the corresponding edges differently in Figure 4 (graph IV). The corresponding unweighted graph is the K_6 graph whose automorphism group is the S_6 group consisting of 6! permutations. It is interesting to ntoe that weighting the edges (graph IV, Figure 4) does not change the automorphism partitioning of the vertices. That is, for graph IV in Figure 4, all six vertices are equivalent. The automorphism group however, consists of 72 permutations characterized by the wreath product group $S_2[S_3]$ described before. Our code gives the correct answer for this graph.

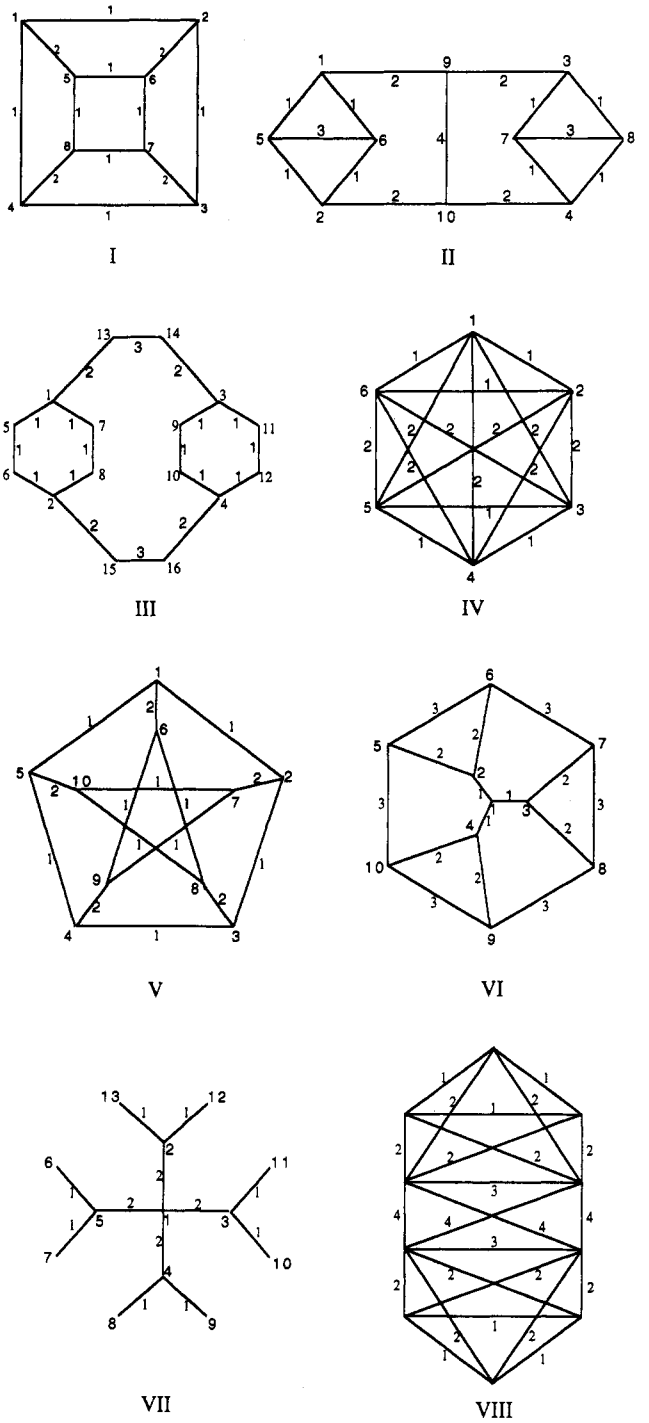

Figure 4. Eight edge-weighted graphs. For the CPU times required to generate the automorphism groups of these graphs, see Table 1.

Table 1. CPU Times Required on an IBM RS/6000-580 Workstation To Generate the Automorphism Groups of Edge-Weighted Graphs in Figure 4

graph	$\lvert\Gamma\rvert$	CPU time (s)	graph	$\lvert\Gamma\rvert$	CPU time (s)
I	16	0.18	V	20	16.45
II	16	0.03	VI	6	0.04
III	8	107.3	VII	384	7.37
IV	72	0.02	VII	288	0.14

Graphs II, III, and VI have been considered before an unweighted forms as regular vertex-transitive graphs of chemical interest. For some of these graphs certain automorphism partitioning techniques do not yield correct results. Our code yields the correct results for edge-weighted graphs with very little CPU time for graphs V and VI. Graph III took 107 s of CPU time, mainly because it has 16 vertices. Furthermore the vertex partioning technique does not reduce

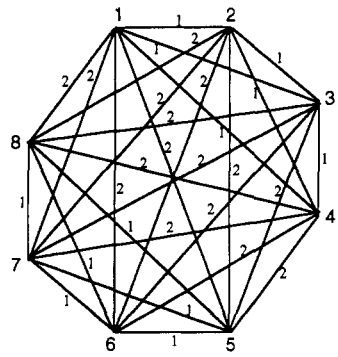

Figure 5. Edge-weighted complete graph on eight vertices.

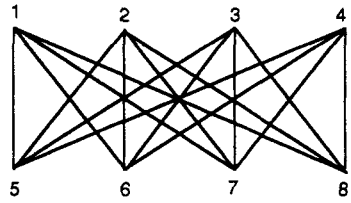

Figure 6. Bipartite graph on eight vertices which becomes isomorphic to the graph in Figure 5 when weights of the latter are changed as follows: $2 \rightarrow 1$ and $1 \rightarrow 0$ (no edge).

the cardinality of the equivalence classes. Therefore more searches are required to generate the automorphism groups.

Weighted tree VII in Figure 4 is interesting in that its automorphism group can be expressed as a wreath product group by a tree pruning algorithm described before. Our code generates all 384 permutations in the automorphism group of this tree in 7.37 s of CPU time. These 384 permutations span the wreath product group $S_4[S_2]$, where S_n is the permutation group comprising $n!$ permutations.

Graph VIII in Figure 4 has ten vertices which are divided into two equivalence classes. The automorphism group is a subgroup of $S_6 \times S_4$. It took 6!4! searches to generate the permutations in the automorphism group. As seen from Table 1, this process took merely 0.14 s of CPU time, which resulted in an automorphism group of 288 permutations. These permutations generate the generalized wreath product group, $S_2[S_3,S_2]$ described before.[23] Hence, our code yields the correct answer for this graph, too.

We also considered a few complete graphs with edge weights. We consider complete graphs K_n for even n. Let us divide the vertices into two sets $\{1, 2, ..., n/2\}\{n/2 + 1, ..., n\}$. All edges within the set are labeled with the same weight 1, while the edges between the vertices of different sets are labeled 2. Such a graph for eight vertices is shown in Figure 5.

Suppose we generate the following mapping between the edge weights.

$$w \rightarrow w'$$

$$1 \rightarrow 0$$

$$2 \rightarrow 1$$

We interpret the 0 edge weight on the right-hand side as no edge. Then, this graph in Figure 5 becomes isomorphic to the bipartite graph in Figure 6. Our code for the automorphism group took only 0.35 s to generate 1152 permutations in the group which comprise the wreath product $S_2[S_4]$. Note that the automorphism group of the unweighted K_8 graph is simply the S_8 group consisting of 8! permutations.

We considered the analogous edge-weighted K_{10} and K_{12} graphs which are isomorphic with the $K_{10,2}$ and $K_{12,2}$ bipartite graphs. Our code took 17.15 s of CPU time to generate 28 800 permutations in the $S_2[S_5]$ wreath product group which is the automorphism group of the edge-weighted K_{10} graph. Our code took 37 min and 31 s of CPU time to generate all 1,036 800 permutations in the automorphism group of the edge-weighted K_{12} graph isomorphic with the $K_{12,2}$ bipartite graph. The group is isomorphic with the wreath product $S_2[S_6]$. This is certainly one of the largest cases considered here in terms of the total number of permutations in the automorphism groups of edge-weighted graphs.

ACKNOWLEDGMENT

This research was supported by the National Science Foundation under Grant CHE 9204999.

REFERENCES AND NOTES

(1) Balasubramanian, K. Computational Techniques for the Automorphism Groups of Graphs *J. Chem. Inf. Comput. Sci.* **1994**, *34*, 621–626.
(2) Balasubramanian, K. Applications of Combinatorics and Graph Theory to Spectroscopy and Quantum Chemistry. *Chem. Rev.* **1985**, *85*, 599–618.
(3) Razinger, M.; Balasubramanian, K.; Munk, M. E. Graph Automorphism Perception Algorithms in Computer-Enhanced Structure Elucidation. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 197–201.
(4) (a) Shelley, C. A.; Woodruff, H. B.; Snelling, C. R.; Munk, M. E. In *Computer-Assisted Structure Elucidation*; Smith, D. H., Ed.; American Chemical Society: Washington, DC, 1977; Vol. 54, Chapter 7, pp 92–107. (b) Gray, N. A. B. *Computer-Assisted Structure Elucidation*; Wiley: New York, 1986; Chapters 7 and 10.
(5) Christie, B. D.; Munk, M. E. The Role of Two-Dimensional Nuclear Magnetic Resonances in Computer-Enhanced Structure Elucidation *J. Am. Chem. Soc.* **1991**, *113*, 3750–3757.
(6) Davis, M. I.; Ellzey, M. L., Jr. A Technique for Determining the Symmetry Properties of Molecular Graphs *J. Comput. Chem.* **1983**, *4*, 267–275.
(7) Shelley, C. A.; Munk, M. E. Computer Perception of Topological Symmetry. *J. Chem. Inf. Comput. Sci.* **1977**, *17*, 110–113.
(8) Shelley, C. A.; Munk, M. E. An Approach to the Assignment of Canonical Tables and Topological Symmetry Perception. *J. Chem. Inf. Comput. Sci.* **1979**, *19*, 247–250.
(9) Herndon, W. C. Canonical Labelling and Linear Notation for Chemical Graphs. In *Chemical Applications of Topology and Graph Theory*; King, R. B., Ed.; Studies in Physical and Theoretical Chemistry; Elsevier: Amsterdam, 1983; Vol. 28, pp 231–242.
(10) Balaban, A. T.; Mekenyan, W.; Bonchev, D. Unique Description of Chemical Structures Based on Hierarchically Ordered Extended Connectivities (HOC Procedures). I. Algorithms for Finding Graph Orbits and Canonical Numbering of Atoms. *J. Comput. Chem.* **1985**, *6*, 538–551.
(11) Balasubramanian, K. Symmetry Groups of Chemical Graphs. *Int. J. Quantum Chem.* **1982**, *21*, 411–418.
(12) Morgan, H. L. Generation of Unique Machine Description for Chemical Structures. A Technique Developed at Chemical Abstracts Service. *J. Chem. Doc.* **1965**, *5*, 107–112.
(13) Razinger, M. Extended Connectivity in Chemical Graphs. *Theor. Chim. Acta* **1982**, *61*, 581–586.
(14) Wipke, W. T.; Dyott, T. M. Stereochemically Unique Naming Algorithm. *J. Am. Chem. Soc.* **1974**, *96*, 4834–4842.
(15) Liu, X.; Balasubramanian, K.; Munk, M. E. Computer-Assisted Graph Theoretical Construction of ^{13}C NMR Signal and Intensity Patterns. *J. Magn. Reson.* **1990**, *87*, 457–476.
(16) King, R. B. The Bonding Topology of Polyhedral Molecules. In *Chemical Applications of Topology and Graph Theory*; King, R. B., Ed.; Studies in Physical and Theoretical Chemistry; Elsevier: Amsterdam, 1983; Vol. 28, pp 108–122.
(17) Randić, M. *Chem. Phys. Lett.* **1976**, *42*, 283.
(18) Randić, M. *Chem. Phys. Lett.* **1974**, *60*, 3920.
(19) Randić, M. Symmetry Properties of Graphs of Interest in Chemistry II. Desargues-Levi Graph. *Int. J. Quantum Chem.* **1979**, *15*, 663–682.
(20) Randić, M.; Davis, M. I. Symmetry Properties of Chemical Graphs VI. Isomerizations of Octahedral Complexes. *Int. J. Quantum Chem.* **1984**, *26*, 69–89.
(21) Nijenhuis, A.; Wilf, H. S. *Combinatorial Algorithms*; Academic Press: New York, 1975.
(22) Balasubramanian, K. The Symmetry of Groups of Non-Rigid Molecules as Generalized Wreath Products. *J. Chem. Phys.* **1980**, *72*, 65.
(23) Balasubramanian, K. Graph Theoretical Characterization of NMR Groups. *J. Chem. Phys.* **1980**, *73*, 3321.