An Efficient Algorithm for Graph Isomorphism

D. G. CORNEIL AND C. C. GOTLIEB

University of Toronto, Toronto, Ontario, Canada*

ABSTRACT. A procedure for determining whether two graphs are isomorphic is described. During the procedure, from any given graph two graphs, the representative graph and the reordered graph, are derived.

The representative graph is a homomorphic image of the original graph; the reordered graph is constructed from the representative graph to be isomorphic to the given graph. Unique labels are assigned to the vertices of both derived graphs. It follows that two representative graphs or two reordered graphs are isomorphic if and only if they are identical. A conjecture states that the representative graphs exhibit the automorphism partitioning of the given graph. The representative graphs form a necessity condition for isomorphism; namely, if the representative graphs are not identical, then the given graphs are not isomorphic. The converse is true for trees and follows from the conjecture for other types of graphs. It is also shown that the reordered graphs form a sufficiency condition for isomorphism; namely, if the reordered graphs are identical, then the given graphs are isomorphic. The converse follows from the conjecture.

The time required to determine both derived graphs depends on a power of n, the order of the given graph. This power is a function of an adjacency property known as the strong regularity of the given graph. For graphs that do not contain a strongly regular transitive subgraph, the power is, at worst, five.

CR CATEGORIES: 3.66, 5.32

KEY WORDS AND PHRASES: graph, graph isomorphism, isomorphism, heuristic procedure, strongly regular graph, automorphisms, transitive graphs, coding of graphs, efficient algorithm, effective algorithm, deterministic algorithm, automorphism partitioning, transitive subgraphs

1. *Introduction*

No efficient deterministic algorithm is known for determining whether two given finite graphs, G_1 and G_2, are isomorphic. An efficient deterministic algorithm is one which guarantees a solution in a time, T, where T is proportional to a constant power of n, the order of the graph. A closely related but inherently more difficult problem is the subgraph isomorphism problem; namely, given two finite graphs, G_1 and G_2, determine whether G_2 is isomorphic to G_1 or to a subgraph of G_1. The graph isomorphism problem and/or the subgraph isomorphism problem arise in such fields as chemistry, information retrieval, linguistics, logistics, switching theory, and network theory. For example, in a chemistry application a chemical compound is represented by its atomic structure diagram (i.e. an undirected, labeled graph). A given compound is matched with the compounds contained in a large file or library [1]. A subgraph isomorphism indicates that the given compound is a sub-

* Department of Computer Science. This paper embodies the results contained in the Ph.D. thesis by D. G. Corneil. Any differences from the results in the thesis are noted. (These are incorporated into a revision sheet for the thesis.)

compound of the library compound under scrutiny; a graph isomorphism indicates that the given compound is already in the library.

Since we are dealing exclusively with finite graphs, a deterministic isomorphism algorithm based upon reordering the nodes is easily given. In this algorithm the nodes of one of the graphs, say G_2, are systematically reordered, and each graph determined by such a reordering is checked for identity with G_1. G_1 is isomorphic to G_2 if and only if G_1 is identical to at least one of the graphs determined by a reordering. The number of node reorderings of G_2 that must be performed is bounded by $n!$. Unger has pointed out that, even using fast present-day computers, a more practical procedure is necessary for dealing with graphs with more than 10 vertices [2].

Since graph isomorphism has many applications, effective computer programs are needed. The lack of an efficient deterministic algorithm has led to inefficient computer procedures, known as heuristic procedures, which do not guarantee an answer in a reasonable time [2, 3, 4]. Heuristic isomorphism procedures attempt to reduce the number of reordering by employing conditions that are necessary for the existence of an isomorphism. These conditions are properties that are invariant under graph isomorphism. For example, no isomorphism between two undirected graphs, G_1 and G_2, may map vertex x of G_1 onto vertex y of G_2 if the degree of x does not equal the degree of y. Using such properties, the upper bound on the number of reorderings of the nodes of G_2 may be reduced. A more detailed description of heuristic procedures is presented in [5].

The graph isomorphism procedure presented in Section 5 of this paper is efficient for all graphs that do not belong to a certain recognizable class. However, since the method is based upon a conjecture, the procedure is *not* deterministic. It terminates with one of the following three statements:

(i) The graphs are isomorphic, with the following isomorphism: . . . .
(ii) The graphs are not isomorphic.
(iii) The graphs form a counterexample to the conjecture.

In the third case it would be necessary to use a deterministic nonefficient heuristic procedure; however, no counterexamples have been discovered.

Only undirected, unlabeled graphs are considered; minor modifications are necessary for other types of graphs (see [5]). In Sections 2 and 3, algorithms which partition the set of vertices of a graph are presented. By using these algorithms, the representative graph and the reordered graph are defined (Section 4). The graph isomorphism algorithm is presented in Section 5; the timing considerations are discussed in Section 6.

Before presenting this material, some definitions are given.

A subgraph,[1] H, of G is a *transitive subgraph of* G if for any two nodes, x, y, of H there exists an automorphism of G mapping x onto y.

A partitioning of the set, V, of vertices of a graph $G(V, E)$ is[2] called the *automorphism partitioning* when the following holds: vertices x and y belong to the same cell if and only if there exists at least one automorphism of G mapping x onto y. Note that the set of vertices in a cell of the automorphism partitioning forms a transitive subgraph.

[1] The set of edges in H is the *restriction* of the set of edges in G to the set of vertices in H.
[2] E is the set of edges in G.

A 2-*strongly regular graph* is an undirected graph, $G(V, E)$, which is not complete and not void such that there exist constants p_{11}^1, p_{12}^1, p_{22}^1, p_{11}^2, p_{12}^2, p_{22}^2 where:

(I) for all $y \in V$, for all $z \in V$ where $(y, z) \in E$:

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

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

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

(II) for all $y \in V$, for all $z \in V$ where $(y, z) \notin E$:

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

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

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

This definition is equivalent to Bose's definition of strongly regular [6]. Condition I indicates that for any two distinct adjacent vertices y and z, there exist exactly p_{11}^1 vertices adjacent to both y and z, exactly p_{12}^1 vertices adjacent to y but not adjacent to z, and exactly p_{22}^1 vertices not adjacent to either y or z. Condition II is similar except that y and z are not adjacent.

The definition of 2-strongly regular may be extended to *h-strongly regular* ($h > 2$) in an obvious way—by specifying, up to isomorphism, all graphs of order h (see [5] for $h = 3$). For purposes of continuity, a regular graph may be called 1-strongly regular.

2. *Terminal Connection Partitioning Algorithm*

In the determination of the representative graph, it is necessary to partition the set of vertices of the given graph. In this section, an algorithm for refining a given partitioning is given. It may be shown that if the given partitioning is invariant under automorphism (and this is always the case for us), the partitioning resulting from this algorithm is also invariant under automorphism. We assume that in the given partitioning, V has been partitioned into i ($i \geq 1$) cells,[3] where the jth cell is denoted V^j ($1 \leq j \leq i$).

ALGORITHM I

Step 1. To each node $y \in V$, associate a list $(a_1, \cdots, a_i)$ where a_j equals the exact number of nodes, $x_l \in V^j$ such that $(y, x_l) \in E$ ($1 \leq j \leq i$). Note that $\sum_{j=1}^{i} a_j = d(y)$, the degree of vertex y.

Step 2. We now define a refinement of the jth cell.

(i) Perform an ordering of the nodes in the cell by examining the lists associated with the nodes of V^j. Order the nodes by lexicographically ordering their lists.

(ii) If all the nodes have the same list, no refinement is done.

(iii) If the lists are not identical, use the ordering of the nodes of V^j to refine V^j as follows: Assume that node y precedes all other nodes of V^j; then the first subcell of V^j consists of node y and all other nodes of V^j which have the same list as y. Remove these nodes from V^j, and from the remaining nodes choose the node that precedes all other remaining nodes.

[3] If i equals 1, then V has not been partitioned.

This node and all other nodes with the same list form the second subcell of V^j. Continue this process until all nodes of V^j belong to a subcell of V^j.

Step 3. Apply step 2 to all i cells. If at least one of the cells is refined, then we reindex all the cells and go to step 1. This reindexing is carried out as follows: Assume that cells 1 to $j-1$ are not refined, but cell j is; cells 1 to $j-1$ retain their previous cell indices. Index the l subcells of cell j as $j, j+1, \cdots, j+l-1$ according to the ordering of subcells defined in step 2. The next cell to be indexed (cell $j+1$ or a subcell of it) is assigned the index $j+l$, and the process continues until all cells have been indexed.

If no cell is refined, the algorithm is finished, and the terminal connection partitioning of the set of nodes of G with respect to the given partitioning has been obtained. Since the refinement requires at least one refinement in order to continue, and the number of cells is bounded by the number of nodes of G (which is finite), the algorithm terminates. If the given partitioning consists of V alone, then the first iteration of Algorithm I merely performs a degree partitioning of V (i.e. two vertices belong to the same cell if and only if their degrees are the same). If G is regular and the given partitioning consists of V alone, then Algorithm I does not refine the partitioning.

Example of Algorithm I. Consider the graph in Figure 1. Assume that we are given the following degree partitioning of V:

Cell index	*Nodes*
I	3, 7, 8, 10
II	6
III	2, 4
IV	1, 5, 9

From step 1 of the algorithm, the lists are:

Cell index	*Node*	*List*
I	3	(3, 0, 2, 0)
	7	(3, 1, 1, 0)
	8	(2, 0, 1, 2)
	10	(2, 0, 1, 2)
II	6	(1, 0, 1, 2)
III	2	(3, 0, 0, 0)
	4	(2, 1, 0, 0)
IV	1	(1, 1, 0, 0)
	5	(1, 1, 0, 0)
	9	(2, 0, 0, 0)

In step 2 the lists are lexicographically ordered, and the nodes are placed in the following order:

Cell index	*Node*	*List*
I	7	(3, 1, 1, 0)
	3	(3, 0, 2, 0)
	8	(2, 0, 1, 2)
	10	(2, 0, 1, 2)
II	6	(1, 0, 1, 2)
III	2	(3, 0, 0, 0)
	4	(2, 1, 0, 0)
IV	9	(2, 0, 0, 0)
	1	(1, 1, 0, 0)
	5	(1, 1, 0, 0)

Cells I, III, and IV are refined. In step 3 the subcells are reindexed, and in step 1 the new lists are assigned as follows:

Cell index	*Node*	*List*
I	7	(0, 1, 2, 1, 0, 1, 0, 0)
II	3	(1, 0, 2, 0, 1, 1, 0, 0)
III	8	(1, 1, 0, 0, 1, 0, 1, 1)
	10	(1, 1, 0, 0, 1, 0, 1, 1)
IV	6	(1, 0, 0, 0, 0, 1, 0, 2)
V	2	(0, 1, 2, 0, 0, 0, 0, 0)
VI	4	(1, 1, 0, 1, 0, 0, 0, 0)
VII	9	(0, 0, 2, 0, 0, 0, 0, 0)
VIII	1	(0, 0, 1, 1, 0, 0, 0, 0)
	5	(0, 0, 1, 1, 0, 0, 0, 0)

Since the lists associated with nodes 8 and 10 are identical and the lists associated with nodes 1 and 5 are identical, the algorithm is finished, and this is the terminal connection partitioning for this graph with respect to the given partitioning.

To represent the refined partitioning of V, we now define Q, the directed quotient graph of a graph G. The terminal connection partitioning algorithm defines the following equivalence relation on V: two nodes belong to the same equivalence class if and only if they belonged to the same initial partition and their terminal lists are identical [5]. Thus we may define the directed *quotient graph*, Q, such that

(i) the set of vertices is the set of integers from 1 to f (f is the number of cells in the terminal connection partitioning);

(ii) the set of directed edges is defined such that if a node, $y \in V^i$, is adjacent to exactly l nodes in V^j (l equals the a_j in the terminal list associated with y), then there is a directed edge of weight l from node i of Q to node j of Q.

Since the vertices are assigned unique integer labels, two quotient graphs are isomorphic if and only if they are identical. The quotient graph, Q, is a homomorphic image of $G(V, E)$ (i.e. the nodes of G are mapped onto the nodes of Q, and the edges of G are mapped onto the edges of Q). Since the nodes in cell V^j form a regular subgraph, say of degree h, of G, there is a directed loop of weight h on vertex j in Q.

Note that the set of rows of B, the adjacency matrix of Q, is the set of lists corresponding to the terminal cells of the partitioning (i.e. the jth row of B is the terminal list associated with each node, $y \in V^j$). B has the properties:

(i) $\sum_{j=1}^{f} b_{ij} = d(x_i)$, for all i, where f is the number of columns in the matrix, and x_i is a vertex in cell i.

(ii) order of $V^i \times b_{ij}$ = order of $V^j \times b_{ji}$, for all i and j.

Example of Quotient Graph. The quotient graph, Q, for the graph given in Figure 1, when the degree partitioning is the given partitioning, is given in Figure 2. The number of arrowheads on the edges indicates the weight of the edge in each direction.

3. *Terminal Quotient Graph*

It was stated in Section 2 that since the partitioning presented to Algorithm I is invariant under automorphism, the partitioning resulting from the algorithm is also invariant under automorphism. Thus any two vertices that belong to a transitive subgraph must belong to the same cell of the refined partitioning. An

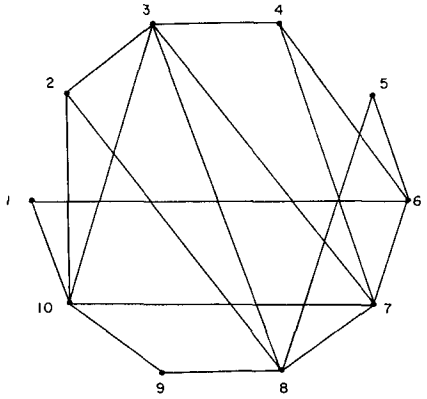

FIG. 1. A graph, G

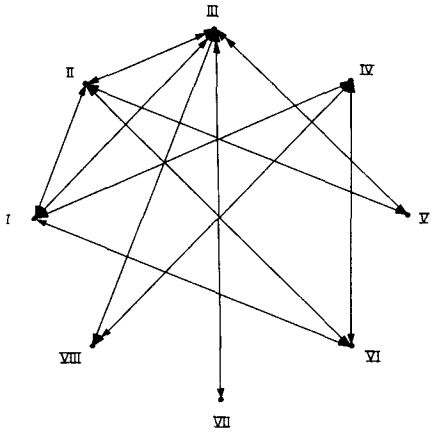

FIG. 2. Q, the quotient graph for G

equivalent statement is the following: the automorphism partitioning is a refinement (possibly trivial) of the partitioning resulting from Algorithm I.

We now present Algorithm II, which utilizes Algorithm I and attempts to determine the maximal transitive subgraphs of the given graph. The algorithm performs a partitioning on the set of vertices and results in a graph which is defined as the terminal quotient graph, Q_T.

ALGORITHM II

Step 1. Perform Algorithm I on V, resulting in the quotient graph, Q. If the given graph is not regular, then the degree partitioning is refined in Algorithm I. If the graph is regular of degree h, then no partitioning of V is achieved, and Q consists of one vertex with a directed loop of weight h on it.

Step 2. Assume that there are i cells ($i \geq 1$). Set k equal to 1 and go to step 3.

Step 3. Examine cell V^k. If there is only one vertex in cell V^k, go to step 7. Assume that there are l ($l > 1$) vertices $x_1, \cdots, x_l$ in V^k. Choose one of these vertices, say x_g, and go to step 4.

Step 4. Perform the following refinement of the existing partitioning (i.e. as represented by Q). Remove x_g from V^k and place it in new cell 1. New indices are assigned to all the old cells of Q as follows: old cell j is assigned the new index $j + 1$ for all j. Effectively, we have altered the given graph G by assigning vertex x_g a unique label and by placing it in a unique cell. The new partitioning is invariant under automorphism for this altered graph. Apply Algorithm I to this new partitioning and obtain the quotient graph Q_{x_g} which will be called the vertex quotient graph for x_g with respect to the given partitioning. Go to step 5.

Step 5. Perform step 4 for all l nodes $x_1, \cdots, x_l$ of cell V^k. If $Q_{x_1} \equiv Q_{x_g}$, for[4] all g ($1 < g \leq l$), then go to step 7; otherwise go to step 6.

[4] In this step we have assumed that the vertices in cell V^k do *not* form an h-strongly regular subgraph ($h \geq 2$). It is necessary to determine the largest h such that the subgraph is h-strongly regular. To do this, Step 4 of Algorithm II is altered so that an h-vertex quotient graph is calculated (i.e. the vertex quotient graph with respect to h given vertices). The h-vertex connection partitioning algorithm, which calculates the h-vertex quotient graph, is defined in the following way: Place each of the chosen h vertices ($x_1, \cdots, x_h$) into an individual cell such that x_i is in cell i; then perform Algorithm I. The resulting quotient graph is called the h-vertex quotient graph $Qx_1, \cdots, x_h$. The following theorem is proved in [5]:

Step 6. If for any two vertices, y, z, in V^k, $Q_y \not\equiv Q_z$, then there can be no automorphism of G that maps y onto z. We may therefore refine the partitioning represented by Q in the following way. The set of vertices in cell V^k is partitioned such that two vertices belong to the same subcell if and only if they possess identical vertex quotient graphs. Now consider each adjacency matrix to be a vector where the $(i + 1)$-th row of the matrix immediately succeeds the ith row for all i. The subcells are then ordered by lexicographically ordering the vectors corresponding to the adjacency matrices of the vertex quotient graphs. This refinement of V^k provides a refinement of the partitioning of V represented by Q. Perform Algorithm I on this new partitioning and reenter Algorithm II at step 2 with the new Q.

Step 7. Replace k with $k + 1$. If k does not equal i, go to step 3. If k equals i, then the algorithm is finished. We now refer to the given quotient graph, Q, as the *terminal quotient graph*, Q_T.

An example of this algorithm will be given in Section 5, where the entire graph isomorphism algorithm is presented.

We now make the following conjecture:

Conjecture. The partitioning resulting from Algorithm II is the automorphism partitioning of V.

Note that the automorphism partitioning is a refinement (we conjecture trivial) of the partitioning from Algorithm II.[5]

In order to deal with the fact that two graphs which possess the same terminal quotient graphs may be nonisomorphic (see Figure 3), we introduce the representative graph.

4. *Representative Graph and Reordered Graph*

Having calculated the terminal quotient graph and the vertex quotient graphs (Algorithm II) we now define the representative graph, G_R.

If all transitive subgraphs of a given graph G are not 2-strongly regular,[6] then G_R, the *representative graph* of G, is immediately derived from Q_T, the terminal quotient graph associated with G. Let Q_{H_i} denote the vertex quotient graph associated with each vertex in the transitive subgraph H_i of G, where the nodes of H_i are mapped onto node i of Q_T for all i. G_R is defined to be the graph Q_T such that

---

THEOREM I. *A graph $G(V, E)$ is h-strongly regular $(2 \leq h < n)$ if and only if*

(i) G is $(h - 1)$-strongly regular;

(ii) In all $(h - 1)$-vertex connection partitionings, two vertices belong to the same cell if and only if their adjacencies to the $(h - 1)$ chosen vertices are identical;

(iii) $Q_{x_1,\cdots,x_{h-1}} \equiv Q_{y_1,\cdots,y_{h-1}}$ if and only if the following holds: $(x_i, x_j) \in E$ if and only if $(y_i, y_j) \in E$ for all i and j.

Thus it is possible to determine m, the maximal strong regularity of a subgraph. If a subgraph is m-strongly regular, then the set of m vertex quotient graphs for all possible choices of m vertices must be calculated. See [5] for further details.

[5] Because of this conjecture, special treatment is required in Algorithm II for h-strongly regular graphs $(h \geq 2)$. This necessity is illustrated by "the exceptional graph of order 26," given in [7]. This graph is nontransitive and 2-strongly regular. Since it is nontransitive, the automorphism partitioning consists of at least two cells; since it is 2-strongly regular, all vertex quotient graphs are identical (Theorem I).

[6] If the transitive subgraph H_j *is* h-strongly regular $(h \geq 2)$ but not $(h + 1)$-strongly regular, then node j of G_R is labeled by the family of h-vertex quotient graphs associated with H_j. See [5] for further details.

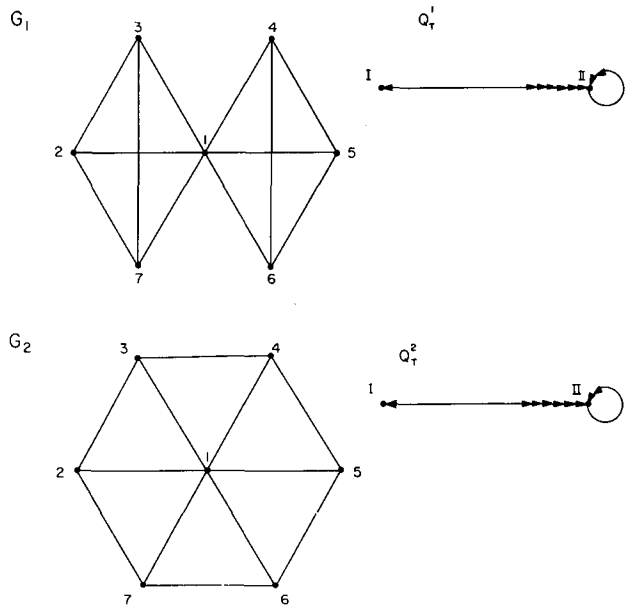

FIG. 3. Nonisomorphic graphs which have identical terminal quotient graphs

node i of G_R is labeled by the vertex quotient graph Q_{H_i} (recall that each vertex in Q_T was labeled with a unique integer).

The representative graphs of the graphs of Figure 3 are shown in Figure 4.

We define two representative graphs to be identical (also isomorphic) if the terminal quotient graphs are identical and if vertices in the representative graphs with the same integer label also have identical vertex quotient graph labels.

THEOREM II. $G_1 \cong G_2 \Rightarrow G_R^1 \equiv G_R^2$.

PROOF. This follows immediately from the fact that the partitionings resulting from Algorithm I and Algorithm II are invariant under automorphism, and therefore invariant under isomorphism.

It is seen that the representative graphs form a necessity condition for isomorphism; namely, if $G_R^1 \not\cong G_R^2$, then $G_1 \not\cong G_2$. Thus we conclude that the graphs in Figure 3 are not isomorphic.

THEOREM III. *If the conjecture is true, then* $G_R^1 \equiv G_R^2 \Rightarrow G_1 \cong G_2$. (Converse of Theorem II.)

PROOF. Consider graph G_3 to be the union of G_1 and G_2. Assume that G_1 and G_2 are connected; otherwise, set G_3 to be the union of the complement of G_1 and the complement of G_2. Let H_i (K_i) denote the transitive (from the conjecture) subgraph of G_1 (G_2) where the nodes of H_i (K_i) are mapped onto note i of Q_T^1 (Q_T^2), for all i. Let $Q_{H_i}^j$ denote the vertex quotient graph (or family of h-vertex quotient graphs) associated with H_i when H_i is a subgraph of G_j. Let D denote the quotient graph of G_1 (and hence G_2) resulting from Algorithm I.

Now apply Algorithm II to G_3. Since $G_R^1 \equiv G_R^2$, $Q_{H_i}^1 \equiv Q_{K_i}^2$ for all i and $Q_T^3 \equiv Q_T^1 \equiv Q_T^2$. Since G_1 and G_2 are disjoint in G_3, $Q_{H_i}^3 \equiv Q_{H_i}^1 \cup D$ and $Q_{K_i}^3 \equiv Q_{K_i}^2 \cup D$ for all i. Thus $Q_{H_i}^3 \equiv Q_{K_i}^3$, and in the partitioning of the set of vertices of G_3 cell i contains $H_i \cup K_i$ for all i. For some cell i choose vertex $x \in V_1$ and vertex $y \in V_2$. Since the given partitioning is assumed to be the automorphism partitioning, we

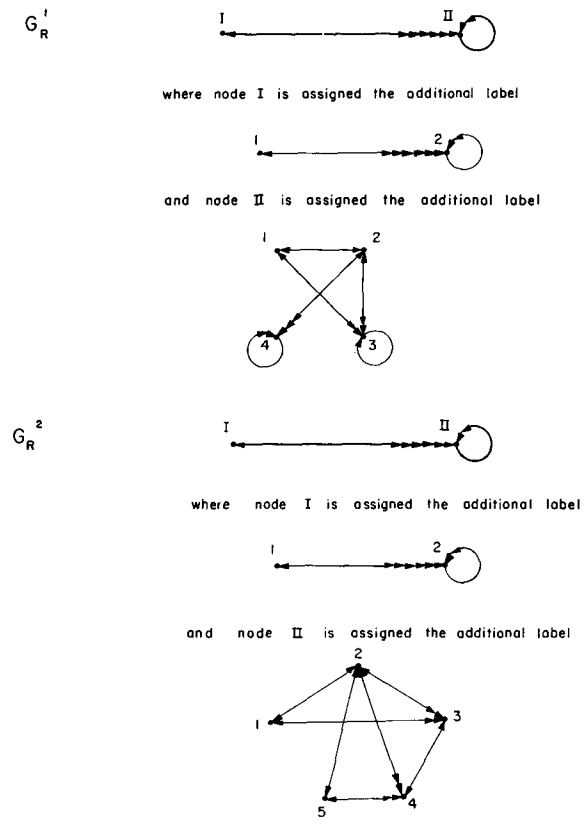

FIG. 4. G_R^1 and G_R^2 for the graphs in Figure 3

know that there exists an automorphism of G that maps x onto y. Since G_1 and G_2 are connected graphs, vertices in V_1 (V_2) may only be mapped onto vertices in V_2 (V_1). Thus this automorphism of G_3 is induced by an isomorphism of G_1 onto G_2, and $G_1 \cong G_2$.[7] It should be noted that the proof requires the validity of Conjecture V.3–1 for graphs of order $2n$ which consist of two disjoint subgraphs of order n.

We now state Theorem IV which shows that the converse of Theorem II holds for trees.

THEOREM IV. *Consider two finite undirected trees T_1 and T_2 with quotient graphs Q^1 and Q^2 (as calculated in step 1 of Algorithm II). Then (i) Q^i ($i = 1$ or 2) is the terminal quotient graph $Q_T{}^i$, and (ii) if $Q^1 \equiv Q^2$ then $T_1 \cong T_2$.*

Other efficient tree isomorphism algorithms not related to our method can be given (e.g. [8, 9]).

The representative graphs form a necessity condition for isomorphism. We now introduce reordered graphs and develop a sufficiency condition. First we present the algorithm for determining the reordered graph, G_r, from the representative

[7] This theorem is not presented in the thesis. All conjectures in the thesis that are related to the isomorphism algorithm have thus been compressed into one essential conjecture. Dr. J. Turner has presented a counterexample to Conjecture V.4–1 of the thesis; however, since this conjecture is not needed, our results are not affected.

graph, G_R. The reordered graph is constructed to be isomorphic to the given graph G. In this algorithm it is assumed that any subgraph whose nodes are mapped onto a single vertex of Q_T is not an h-strongly regular graph ($h \geq 2$). For these types of graphs see [5].

ALGORITHM III

Step 1. Does the number of vertices in Q_T equal the number of nodes in G? If not, go to step 2; otherwise, Q_T represents a unique reordering of the nodes of G, the adjacency matrix of G_r is identical to the adjacency matrix of Q_T, and the algorithm is finished.

Step 2. Let V^i denote the ith cell of the partitioning of V represented by Q_T. Choose the cell with the lowest index, say j, such that the order of $V^j \neq 1$. From V^j arbitrarily choose a vertex, say x. Perform step 4 of Algorithm II to construct the vertex quotient graph Q_x (this vertex quotient graph is formed with respect to the current partitioning of V). Perform steps 2 to 7 of Algorithm II to refine the partitioning represented by Q_x, obtaining a new terminal quotient graph Q_T. Go to step 1.

Example of Algorithm III. In Figure 3, G_2 and Q_T^2 are given. The homomorphic mapping of V onto Q_T^2 is

$$(1, (2, 3, 4, 5, 6, 7)) \Rightarrow (\text{I}, \text{II}).$$

Since the order of $Q_T^2 = 2$ and the order of $V = 7$, we perform step 2. Here we set $j = \text{II}$, and choose a vertex, say 4. Q_4 is calculated (see the vertex quotient graph label assigned to vertex II of G_R in Figure 4). This partitioning is not refined in step 2. The mapping of V onto the new Q_T is

$$(4, 1, (3, 5), (2, 6), 7) \Rightarrow (\text{I}, \text{II}, \text{III}, \text{IV}, \text{V}).$$

In step 1, since the order of $Q_T = 5$ and the order of $V = 7$ we return to step 2, set $j = \text{III}$ and choose a vertex, say 5. Q_5 is calculated (see Figure 5), no refinement is achieved, and we perform step 1. The mapping of V onto Q_5 is

$$(5, 4, 1, 3, 6, 2, 7) \Rightarrow (\text{I}, \text{II}, \text{III}, \text{IV}, \text{V}, \text{VI}, \text{VII}).$$

Since the order of Q_T is 8, which is the order of V, the algorithm is finished and G_r is given in, Figure 6.

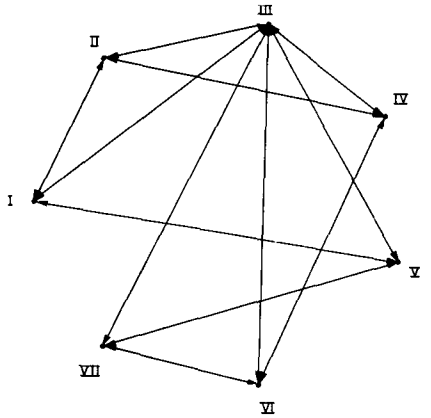

FIG. 5. The vertex quotient graph, Q_5, for G_2 of Figure 3

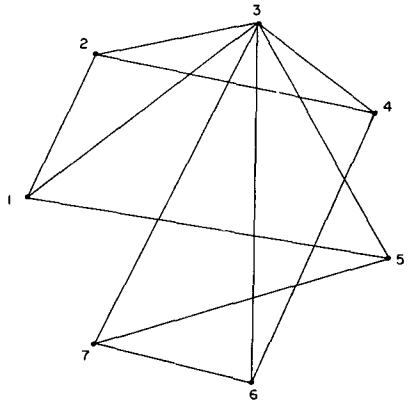

FIG. 6. G_r, the reordered graph, for G_2 of Figure 3

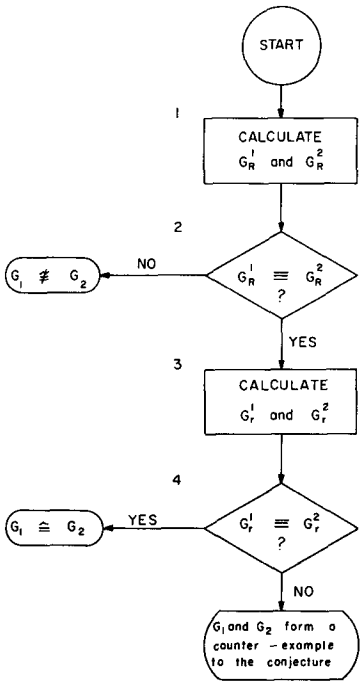

FIG. 7. Algorithm IV, the graph isomorphism algorithm

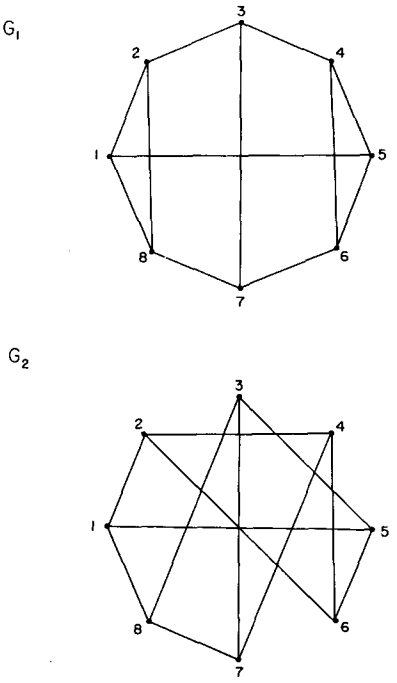

FIG. 8. G_1 and G_2, two graphs to be tested for isomorphism

We now present a theorem regarding reordered graphs.

THEOREM V. $G_r^1 \equiv G_r^2 \Rightarrow G_1 \cong G_2$.

PROOF. This is obvious since $G_r^1 \cong G_1$ and $G_r^2 \cong G_2$.

It is seen that the reordered graphs form a sufficiency condition for isomorphism; namely, if $G_r^1 \equiv G_r^2$, then $G_1 \cong G_2$. The converse of this theorem is true if the Conjecture is true (see [5]). Thus if the Conjecture is true, then the reordered graph is a canonical form for the equivalence class of graphs isomorphic to the given graph.

5. *Graph Isomorphism Algorithm*

ALGORITHM IV

This algorithm is presented in the form of a flowchart in Figure 7. It may be shown that if the Conjecture is true, then $G_R^1 \equiv G_R^2 \Rightarrow G_r^1 \equiv G_r^2$. In the isomorphism algorithm, it is a violation of this statement that would indicate that a counter-example had been found to the Conjecture.

Example of Algorithm IV. We are given the graphs, G_1 and G_2, presented in Figure 8. In block 1 of Algorithm IV, the representative graphs, G_R^1 and G_R^2, are derived. The execution of Algorithm II on G_1 will be followed in detail.

After step 1 of Algorithm II, Q consists of one vertex with a directed loop of weight 3 on it. In step 2, k is set equal to 1. In steps 3, 4, and 5, the vertex quotient graphs Q_i $(1 \leq i \leq 8)$ are calculated. In step 6 it is seen that $Q_1 \equiv Q_5$; $Q_2 \equiv Q_4 \equiv Q_6 \equiv Q_8$; $Q_3 \equiv Q_7$. The set of vertices is partitioned and the subcells are ordered such that cell {1, 5} precedes cell {3, 7} which precedes cell {2, 4, 6, 8}. Algorithm I does not refine this partitioning so that we now

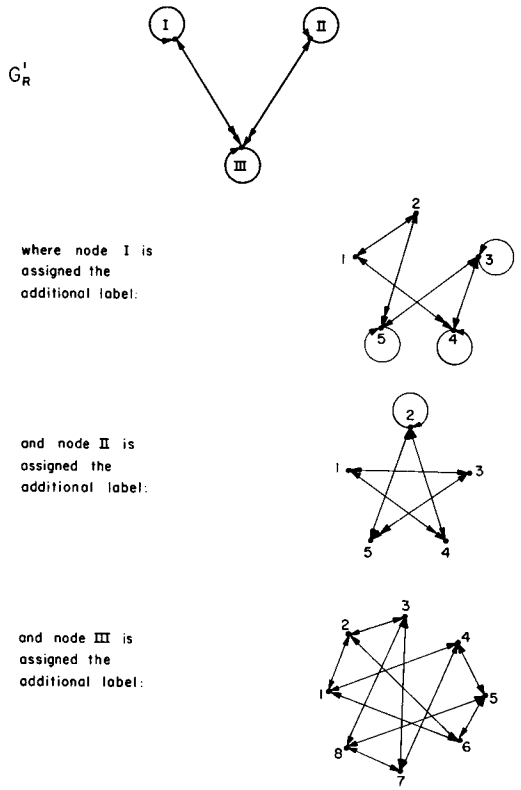

FIG. 9. G_R , the representative graph for both G_1 and G_2

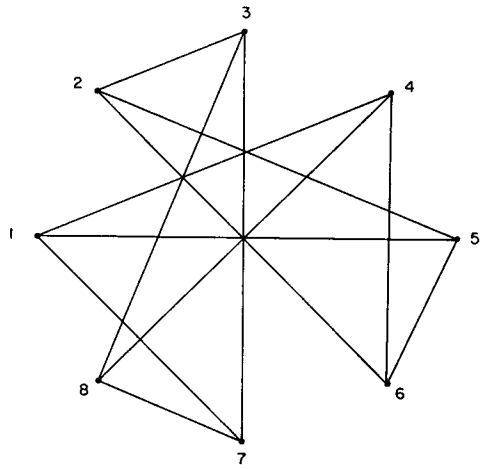

FIG. 10. G_r , the reordered graph for both G_1 and G_2

have a new quotient graph Q. (Q is the same as Q_T given in Figure 9.) Algorithm II is reentered at step 2. It is found that no further refinement is achieved.[8]

The representative graphs for both G_1 and G_2 are given in Figure 9. Since $G_R^1 \equiv G_R^2$, the yes exit of block 2 is followed. In block 3, G_r^1 and G_r^2 are determined (see Figure 10). Since $G_r^1 \equiv G_r^2$, the yes exit of block 4 is taken and we conclude that G_1 is isomorphic to G_2 . One possible

[8] In fact, we know of no example in which a refinement is achieved at this point.

isomorphism between G_1 and G_2 is $\sigma(4, 2, 1, 8, 7, 3, 5, 6)$ where node j of G_1 is mapped onto node $\sigma(j)$ of G_2 (e.g. node 4 of G_1 is mapped onto node 8 of G_2). Further examples of the algorithm are presented in [5].

6. *Timing Considerations*

In this section we examine the timing requirement for Algorithms I and IV; the other algorithms are examined in [5]. We determine the dependence of the processing time, T, on n by examining each step of the algorithm and estimating the number of machine cycles needed to perform this step. The following assumptions are made: the binary adjacency matrices are packed into the machine words; δ_1 machine cycles are needed to interrogate an element of a packed binary matrix; δ_2 cycles are needed to compare two words; δ_3 cycles are needed for an integer addition; δ_4 cycles are needed for a word replacement; the times for all other operations, particularly indexing operations for controlling loops and for modifying fetch instructions, are negligible. For each step (and thus for the algorithm), the largest term is found for each of δ_1 , δ_2 , δ_3 , and δ_4 . This analysis is similar to that in [10].

In the analysis of Algorithm I, the following terminology is used: t denotes the iteration that is being performed; in the t-th iteration, the set of nodes has been previously partitioned into $f(t)$ cells; f is the order of the quotient graph; $n_j{}^t$ denotes the number of nodes in cell V^j in the t-th iteration ($j = 1, \cdots, f(t)$); there are exactly $h_j{}^t$ different lists assigned to the nodes in V^j in the t-th iteration; $N = \sum_t n^2$; $F = \sum_t f(t)$. The number of machine cycles required by the implemented version of the algorithm is

$$T = N\delta_1 + \left(N + \sum_t f(t) \sum_j n_j{}^t\left(3 - \frac{1}{h_j{}^t}\right)\right)\delta_2 + 2(N + nF)\delta_3 + (f^2 + N + nF)\delta_4 + \text{terms which depend on lower powers of the parameters.}$$

An upper bound on this expression is

$$n^3(\delta_1 + \tfrac{5}{2}\delta_2 + 3\delta_3 + \tfrac{3}{2}\delta_4) + \text{terms which depend on lower powers of } n.$$

(This bound is unreachable.)

For Algorithm IV, the maximum number of machine cycles required for graphs that do not contain 2-strongly regular transitive subgraphs is [5]:

$$n^5(\delta_1 + \tfrac{5}{2}\delta_2 + 3\delta_3 + \tfrac{3}{2}\delta_4) + \text{terms which depend on lower powers of } n.$$

As an example of the validity of these timing expressions, we used the IBM 7094-II to examine the predicted timings and the observed timings for random graphs and polygons. For isomorphic random graphs the processing time depended on n^2. For density of edges = 0.5 and $n = 20$, the predicted time is 0.00363 min, and the observed time is 0.00447 min; for $n = 60$, the predicted time is 0.0323 min, and the observed time is 0.0330 min. The predicted time for isomorphic polygons depends on n^4; this family of graphs seems to possess the worst dependence on n for graphs that do not contain 2-strongly regular transitive subgraphs. For $n = 10$, the predicted time is 0.0346 min, and the observed time is 0.0453 min; for $n = 40$, the predicted time is 7.77 min, and the observed time is 9.97 min; for $n = 60$, the predicted time is 38.7 min.

For graphs that contain a transitive h-strongly regular subgraph [not $(h + 1)$-

strongly regular ($h \geq 2$)], the upper bound on the timing for the graph isomorphism algorithm depends on n^{5+h}. Since the upper bound for h may be n, our isomorphism algorithm is inefficient for families of graphs whose strong regularity is a function of n.

For other graphs and in particular for graphs encountered in usual applications, the algorithm is efficient and, subject to the Conjecture, highly effective.

ACKNOWLEDGMENTS. The authors are indebted to Professor A. B. Lehman of the University of Toronto for the clarification resulting from his careful criticism of both the thesis and this paper. Professor Lehman pointed out how the representative graphs form a necessity condition and how the reordered graphs form a sufficiency condition for isomorphism. The authors also wish to thank the National Research Council of Canada for financial assistance and to acknowledge the helpful suggestions made by Dr. J. Turner of the Stanford Research Institute.

REFERENCES

1. LYNCH, M. F. Storage and retrieval of information on chemical structures by computer. *Endeavour 27*, 101 (May 1968), 68–73.
2. UNGER, S. H. GIT—a heuristic program for testing pairs of directed line graphs for isomorphism. *Comm. ACM 7*, 1 (Jan. 1964), 26–34.
3. SUSSENGUTH, E., JR. A graph theoretical algorithm for matching chemical structures. *J. Chem. Doc. 5*, 1 (Feb. 1965), 36–43.
4. BÖHM, C., AND SANTOLINI, A. A quasi-decision algorithm for the p-equivalence of two matrices. *ICC Bull. 3*, 1 (1964), 57–69.
5. CORNEIL, D. G. *Graph Isomorphism.* Ph.D. thesis, U. of Toronto, Canada, 1968.
6. BOSE, R. C. Strongly regular graphs, partial geometries, and partially balanced designs. *Pacific J. Math. 13* (1963), 389–420.
7. GOETHALS, J. M., AND SEIDEL, J. J. Orthogonal matrices with zero diagonal. *Canad. J. Math. 19* (1967), 1001–1010.
8. SMOLENSKII, Y. A. A method for the linear recording of graphs. *USSR Comput. Math. and Math. Phys. 2* (1963), 396–397.
9. BUSACKER, R., AND SAATY, T. *Finite Graphs and Networks—An Introduction with Applications.* McGraw-Hill, New York, 1965, 196–199.
10. GOTLIEB, C. C., AND CORNEIL, D. G. Algorithms for finding a fundamental set of cycles for an undirected linear graph. *Comm. ACM 10*, 12 (Dec. 1967), 780–783.

RECEIVED JULY, 1968; REVISED SEPTEMBER, 1968