ABSTRACT

A new algorithm is presented for the related problems of canonically labelling a graph or digraph and of finding its automorphism group. The automorphism group is found in the form of a set of less than \( n \) generators, where \( n \) is the number of vertices. An implementation is reported which is sufficiently conserving of time and space for it to be useful for graphs with over a thousand vertices.

1. INTRODUCTION

Let \( V \) be the finite set \( \{1, 2, \ldots, n\} \). Define \( \mathcal{G}(V) \) to be the set of all (labelled) graphs with vertex set \( V \). Let \( S_n \) be the symmetric group acting on \( V \). For \( G \in \mathcal{G}(V) \) and \( g \in S_n \), define \( G^g \in \mathcal{G}(V) \) to be the graph in which vertices \( v^g \) and \( w^g \) are adjacent exactly when \( v \) and \( w \) are adjacent in \( G \). The automorphism group of \( G \), \( \text{Aut}(G) \), is the group \( \{ g \in S_n | G^g = G \} \).

The canonical label problem is to find a map \( \text{canon}: \mathcal{G}(V) \rightarrow \mathcal{G}(V) \) such that for \( G \in \mathcal{G}(V) \) and \( g \in S_n \),

1. \( \text{canon}(G) \) is isomorphic to \( G \), and
2. \( \text{canon}(G^g) = \text{canon}(G) \).

Note that there may be many functions \( \text{canon} \) satisfying (1) and (2).

If \( G, H \in \mathcal{G}(V) \), we see that \( G \) and \( H \) are isomorphic if and only if \( \text{canon}(G) = \text{canon}(H) \).

In this paper we present a new algorithm for computing \( \text{canon}(G) \) which will also find a set of fewer than \( n \) automorphisms which generate \( \text{Aut}(G) \). With only minor modifications which we will indicate, the algorithm is equally applicable to digraphs. Undefined graph theoretic or group theoretic concepts can be found in [1] or [7] respectively.

2. EQUITABLE PARTITIONS

Let \( V = \{1, 2, \ldots, n\} \). A partition of \( V \) is a collection \( \pi \) of disjoint non-empty subsets of \( V \) whose union is \( V \). The elements of \( \pi \) are called its cells. An ordered partition of \( V \) is a sequence \( (C_1, C_2, \ldots, C_k) \) for which \( \{C_1, C_2, \ldots, C_k\} \) is a partition. The sets of all partitions of \( V \), and of all ordered partitions of \( V \) will be denoted by \( \Pi(V) \) and \( \Pi(V) \) respectively.
Define $\Pi^*(V) = \Pi(V) \cup \Pi(V)$. Let $\pi_1, \pi_2 \in \Pi^*(V)$. We write $\pi_1 \leq \pi_2$ ($\pi_1$ is finer than $\pi_2$, $\pi_2$ is coarser than $\pi_1$) if every cell of $\pi_1$ is contained in some cell of $\pi_2$. If both $\pi_1 \leq \pi_2$ and $\pi_2 \leq \pi_1$, we write $\pi_1 \simeq \pi_2$. If $\pi \in \Pi^*(V)$, the number of cells of $\pi$ is denoted by $|\pi|$. $\pi$ is called discrete if $|\pi| = n$.

Let $\pi \in \Pi^*(V)$ and $g \in S_n$. Then $\pi^g \in \Pi^*(V)$ is formed by replacing each cell $C \in \pi$ by $C^g$. If $\pi = \pi^g$, $g$ is said to fix $\pi$. Denote by $\pi \vee g$ the finest partition of $V$ which is coarser than $\pi$ but fixed by $g$. The existence of $\pi \vee g$ follows from the fact that $(\Pi(V), \leq)$ is a lattice [3].

Choose a fixed $G \in \mathcal{G}(V)$. If $W \subseteq V$ and $v \in V$, the number of vertices in $W$ which are adjacent to $v$ will be denoted by $d(v, W)$. Let $\pi \in \Pi^*(V)$. $\pi$ is said to be equitable (for $G$) if for any $C_1, C_2 \in \pi$ and $v_1, v_2 \in C_1$ we have $d(v_1, C_2) = d(v_2, C_2)$. For an arbitrary $\pi$, the coarsest equitable partition which is finer than $\pi$ will be denoted by $\xi(\pi)$. Similarly, $\theta(\pi)$ denotes the partition whose cells are the orbits of the subgroup of $\text{Aut}(G)$ which fixes $\pi$. The proof of the following lemma can be found in [3].

**Lemma 1.** Let $\pi \in \Pi(V)$. Then

(i) $\theta(\pi) \leq \xi(\pi)$,

(ii) $\theta(\pi)$ is equitable, and

(iii) if $\pi$ is equitable, and $n - |\pi| \leq 5$, the smallest cells of $\pi$ of size $\geq 2$ are cells of $\theta(\pi)$. (Not true if $G$ is a digraph.) □

Corneil proved in [2] that for any $\pi$, $\theta(\pi) = \xi(\pi)$ if $G$ is a tree. This can be generalised to uni-cyclic graphs and many others. See [3] for further details.

Algorithms for computing $\xi(\pi)$ have been used many times in graph isomorphism programs ([2], [5], [6]). For our own purposes, however, the following system appears to be more efficient. Let $\pi \in \Pi(V)$ and let $a$ be a subset of $\pi$.

**Algorithm 1:** Compute $\widetilde{\pi} = \mathcal{R}(G, \pi, a)$

(1) $\widetilde{\pi} + \pi$

(2) If $a = \emptyset$ or $\widetilde{\pi}$ is discrete, stop.

Choose any non-null subset $\beta$ of $a$.

$\alpha + \alpha \setminus \beta$, $i \leftarrow 1$

(Suppose $\widetilde{\pi} = \{C_1, C_2, \ldots, C_k\}$ and $\beta = \{W_1, W_2, \ldots, W_r\}$.)

(3) Partition $C_i$ into subsets $D_1, D_2, \ldots, D_s$ according to the vectors

$(d(v, W_1), d(v, W_2), \ldots, d(v, W_r))$ for $v \in C_i$.

$\widetilde{\pi} + \widetilde{\pi} \cup \{D_1, D_2, \ldots, D_s\} \setminus \{C_i\}$
\(a + a \cup \{D_2, \ldots, D_s\} \quad (\text{if } s \geq 2)\)

\(i \leftarrow i + 1\)

If \(i \leq k\) go to (3); otherwise go to (2).

**THEOREM 1.** For any \(\pi \in \Pi(V)\), \(\mathcal{R}(G, \pi, \pi) \preceq \xi(\pi)\).

**Proof:** (a) In step (3) of the algorithm, \(|a|\) is not increased unless \(\mathcal{W}\) is made finer. Since \(|a|\) is reduced in step (2), the algorithm is sure to terminate.

(b) By definition, \(\xi(\pi) \leq \pi\). Suppose that before some execution of step (3) we have \(\xi(\pi) \leq \mathcal{W}\). Since \(\beta \subseteq \mathcal{W}\), each element of \(\beta\) is a union of cells of \(\xi(\pi)\). Hence we will also have \(\xi(\pi) \leq \mathcal{W}\) after execution of step (3). Therefore \(\xi(\pi) \leq \mathcal{R}(G, \pi, \pi) \leq \pi\).

(c) Suppose that \(\mathcal{R}(G, \pi, \pi)\) is not equitable. Then for some \(C_1, C_2 \in \mathcal{R}(G, \pi, \pi)\), there are \(v, w \in C_1\) such that \(d(v, C_2) \neq d(w, C_2)\). Since \(\mathcal{W}\) is made successively finer by the algorithm, \(v\) and \(w\) must always be in the same cell of \(\mathcal{W}\).

At step (1), \(C_2\) is contained in some cell of \(a\). Hence \(C_2\) must sometime be contained in an element \(W\) of \(\beta\).

(d) Since \(v\) and \(w\) are never separated, \(d(v, W) = d(w, W)\). Hence there is a cell \(C_3\) of \(\mathcal{R}(G, \pi, \pi)\) other than \(C_2\) for which \(d(v, C_3) \neq d(w, C_3)\). Since \(C_2\) and \(C_3\) are different cells of \(\mathcal{R}(G, \pi, \pi)\), they must be separated sometime in step (3). At least one of them, say \(C_2\), will then be contained in a new element of \(a\).

(e) Since the argument in (d) can be repeated indefinitely, the algorithm never terminates, contradicting (a). Hence \(\mathcal{R}(G, \pi, \pi)\) is equitable, and so \(\mathcal{R}(G, \pi, \pi) = \xi(\pi)\).

A considerable advantage which Algorithm 1 has over previous algorithms is that in some important situations, \(\alpha\) can be a proper subset of \(\pi\). Suppose \(\pi_1\) is an equitable partition of \(V\) coarser than \(\pi\). Let \(\alpha \subseteq \pi\) be such that for any \(D \in \pi_1, C \subseteq D\) for at most one \(C \in \pi \setminus \alpha\). Then it can be proved (see [3]) that \(\mathcal{R}(G, \pi, \alpha) = \xi(\pi)\).

When implemented on a computer, Algorithm 1 becomes an operation on ordered partitions, and produces an ordered result. We will presume an implementation such that for any \(g \in S_n\), \(\mathcal{R}(G^g, \pi^g, \alpha^g) = \mathcal{R}(G, \pi, \alpha)^g\).

3. **THE BASIC STRUCTURE OF THE ALGORITHM**

Our algorithm for computing \(\text{canon}(G)\) and generators for \(\text{Aut}(G)\) is based on a depth-first search through a tree whose nodes are equitable ordered partitions of \(V\).
The first node, or root, is the equitable partition $\mathcal{R}(G, (V), (V))$, where $(V)$ is the ordered partition with one cell. Suppose that $\pi = (C_1, C_2, \ldots, C_r)$ is an arbitrary node of the search tree. If $\pi$ is discrete, it is an end-node of the tree and we will call it a terminal partition. If $\pi$ is not discrete, let $C_1$ be the first cell of $\pi$ with the smallest size $\geq 2$. For $v \in C_1$ define $\pi \cdot v = (C_1, \ldots, C_i \backslash \{v\}, \ldots, C_r, \{v\})$. The successors of $\pi$ in the search tree are the partitions $\mathcal{R}(G, \pi \cdot v, \{v\})$, for $v \in C_1$, which are equitable by the remark following Theorem 1.

Let $X$ be the set of terminal partitions in the search tree. Given any $\tau \in X$, we can form the graph $G^\tau$ by labelling the vertices of $G$ in the order they appear in $\tau$. Define the equivalence relation $\sim$ on $X$ by $\tau_1 \sim \tau_2$ if and only if $G^{\tau_1} = G^{\tau_2}$. It is easy to show ([3] or [4]) that for any fixed $\varepsilon \in X$, $\text{Aut}(G) = \{g|\tau = g^\varepsilon, \tau \in X, \tau \sim \varepsilon\}$. Furthermore, by comparing the adjacency matrices $\varepsilon \in X$, we can define an order on $G(V)$ and define $\text{canon}(G) = \max \{G^\tau|\tau \in X\}$.

Since $|X|$ is a multiple of $|\text{Aut}(G)|$ it is usually not feasible to generate the entire search tree. This problem is largely overcome by the methods described in [3] or [4] which use discovered automorphisms of $G$ to eliminate sections of the search tree from consideration. However, while these methods will reduce the size of each equivalence class of terminal partitions to a manageable size (usually $n$ or less), they will not reduce the number of classes. One way of doing this is to define a function $\Lambda(G, \pi)$ for each $G \in G(V)$ and $\pi \in \Pi(V)$ such that $\Lambda(G^\varepsilon, \pi^\varepsilon) = \Lambda(G, \pi)$ for any $g \in S_n$. An example of such a function would be the number of edges of $G$ whose endpoints are both in the same cell of $\pi$. Now suppose that $\tau \in X$ and that $\pi_1 \geq \pi_2 \geq \cdots \geq \pi_k = \tau$ is the sequence of equitable partitions from the root of the search tree to $\tau$. Then $\tau$ can be associated with the sequence $\Lambda(\tau) = (\Lambda(G, \pi_1), \Lambda(G, \pi_2), \ldots, \Lambda(G, \pi_k))$. Clearly $\tau_1$ and $\tau_2$ cannot be equivalent unless $\Lambda(\tau_1) = \Lambda(\tau_2)$. Further efficiency can be achieved by ordering the vectors $\Lambda(\tau)$ lexicographically and redefining $\text{canon}(G)$ to be $\max \{G^\tau|\tau \in X, \Lambda(\tau) = \Lambda^*\}$, where $\Lambda^* = \max \{\Lambda(\tau)|\tau \in X\}$. By this means we can eliminate sections of the search tree which cannot contain either new automorphisms or $\text{canon}(G)$.

In some cases, additional means of acceleration is provided by Lemma 1 (iii). If $\pi$ is an equitable partition in the search tree and $n - |\pi| \leq 5$, all the terminal partitions descended from $\pi$ are equivalent.

4. THE ALGORITHM

ALGORITHM 2: Compute generators for $\text{Aut}(G)$, where $G$ is a graph or digraph, and optionally compute $\text{canon}(G) = G^\beta$. 
Notes:  
(i) Lower case Greek letters represent partitions. The variables $C_1, C_2, \ldots$ represent sets.
(ii) The minimum of an empty set is defined to be $\infty$, where $\infty$ is a number larger than any number it is compared to.
(iii) If $x$ is a permutation (or partition), $\hat{u}(x)$ denotes the set consisting of the smallest element of each cycle (or cell) of $x$, and $\Phi(x)$ is the set of elements in trivial cycles (or cells of size 1) of $x$.
(iv) $lab$ and $dig$ are boolean variables. $lab = true$ if canon $(G)$ is required. $dig = true$ if $G$ is a digraph, or a graph with loops.

(1) $k \leftarrow l$, size $\leftarrow l$, $h \leftarrow 0$, index $\leftarrow 0$
$\emptyset$ + discrete partition of $V$
$\pi_1 \leftarrow \mathcal{A}(G, \{V\}, \{V\})$
If $dig$ or $n - |\pi_1| \geq 6$, $q \leftarrow 2$, else $q \leftarrow 1$.
If $\pi_1$ is discrete go to (20).
$C_1 \leftarrow$ first cell of $\pi_1$ with smallest size $\geq 2$
$e_1 \leftarrow 0$, $v_1 \leftarrow \min C_1$, $w_1 \leftarrow v_1$

(2) $k \leftarrow k + 1$
$\pi_k \leftarrow \mathcal{A}(G, \pi_{k-1} \circ \{v_{k-1}\}, \{v_{k-1}\})$
$z \leftarrow \Lambda(G, \pi_k)$
If $\pi_k$ is not discrete, $e_k \leftarrow 0$ and $C_k \leftarrow$ first cell of $\pi_k$ with smallest size $\geq 2$.
If $h = 0$, go to (6).
If $hx = k - 1$ and $z = x_k$, $hx \leftarrow k$
If not $lab$, go to (4).
If $hy \neq k - 1$, go to (3).
$qy \leftarrow z - y_k$
If $qy = 0$, $hy \leftarrow k$.

(3)' If $qy > 0$, $y_k \leftarrow z$.

(4) If $hx = k$ or ($lab$ and $qy \geq 0$), go to (5).
$k \leftarrow q - 1$
Go to (9).
(5) If \( \pi_k \) is discrete, go to (7).
\[
v_k = \min C_k
\]
If \( h = 0 \), \( w_k \leftarrow v_k \).
If \( d \leq g \) or \( n - |\pi_k| \geq 6 \), \( q \leftarrow k + 1 \).
Go to (2).

(6) If \( lab \), \( y_k \leftarrow z \).
\[
x_k \leftarrow z
\]
Go to (5).

(7) If \( h < q \), go to (15).
Compute the permutation \( g \) such that \( \pi^G = \pi_k \).

(8) \((g \in \text{Aut}(G)): \) Write \( g \) if desired, and store \((\phi(g), \Omega(g))\) if room is available.
\[
\theta \leftarrow \theta \cup g, k \leftarrow h
\]

(9) If \( k = 0 \), stop.
If \( k > h \), go to (13).
\[
h = \min \{k, h\}
\]

(10) If \( v_k = w_k \) are in the same cell of \( \theta \), index \( + \) index \( + \) 1.
\[
v_k \leftarrow \min \{v \in C_k | v > v_k \}
\]
If \( v_k = \infty \), go to (12).
If \( v_k \neq \Omega(\theta) \), go to (10).

(11) \( q \leftarrow \min \{q, k + 1\}, hx \leftarrow \min \{hx, k\} \)
If not \( lab \), go to (12).
\[
hb \leftarrow \min \{hb, k\}
\]
If \( hy < k \), go to (2).
\[
hy \leftarrow k, qx \leftarrow 0
\]
Go to (2).

(12) size \( + \) size \( \times \) index
index \( + \) 0, \( k \leftarrow k - 1 \)
Go to (9).

(13) If \( e_k = 0 \), go to (14).
\[
e_k \leftarrow 1
\]
For any stored pairs \((\phi(g), \Omega(g))\) such that \(\{v_1, v_2, \ldots, v_{k-1}\} \subseteq \phi(g)\),
set \( C_k \leftarrow C_k \cap \Omega(g) \).
(14) \( v_k + \min \{ v \in C_k \mid v > v_k \} \)
If \( v_k = \infty \), set \( k + k - 1 \) and go to (9).
Go to (11).

(15) If \( h = 0 \), go to (20).
If \( h \neq k \), go to (16).
Compute the permutation \( g \) such that \( g^G = \pi_k \).
If \( g \in \text{Aut}(G) \), go to (8).

(16) If \( q_y < 0 \) or not \( \text{lab} \), go to (18).
If \( q_y > 0 \), go to (17).
If \( G^G = G^\pi_k \), go to (19).
If \( G^G > G^\pi_k \), go to (18).

(17) \( b + \pi_k, hy + k, hb + k, y_{k+1} + \infty, q_y + 0 \)

(18) \( k + q - 1 \)
Go to (9).

(19) \( k + hb \)
If \( k \neq h \), go to (9).
Compute the permutation \( g \) such that \( g^G = \pi_k \).
Go to (8).

(20) \( h + k, hx + k, x_{k+1} + \infty, e + \pi_k \)
\( k + k + 1 \)
If not \( \text{lab} \), go to (9).
\( b + \pi_{k-1}, hy + k + 1, hb + k + 1, y_{k+2} + \infty, q_y + 0 \)
Go to (9).

Let \( G \) be a graph or a digraph and let \( A = \text{Aut}(G) \). If \( W \subseteq V \), \( \Lambda_W \) denotes the point-wise stabiliser of \( W \) in \( A \). Consider the instant when the first line of step (12) has been executed for a particular value of \( k \). Define \( \theta^{(k-1)} \), \( \text{index}^{(k-1)} \), \( \text{size}^{(k-1)} \) to be the current values of \( \theta \), \( \text{index} \) and \( \text{size} \), and let \( \Sigma^{(k-1)} \) be the set of all elements of \( \text{Aut}(G) \) found by this stage.

**THEOREM 2.**
(1) Let \( K \) be the value of \( k - 1 \) at the start of step (20). Define \( A^{(0)} = A \) and \( A^{(k)} = A\{w_1, \ldots, w_k\} \) for \( 1 \leq k \leq K \). Then for \( 0 \leq k \leq K \),

(i) \( \text{size}^{(k)} = |A^{(k)}| \)
(ii) \( \text{index}^{(k)} = |A^{(k)}|/|A^{(k+1)}| \) \((k < K)\)
(iii) the cells of \( \theta^{(k)} \) are the orbits of \( A^{(k)} \)
(iv) \( \Sigma^{(k)} \) generates \( A^{(k)} \)
(v) \( |\Sigma^{(k)}| \leq n - \lambda_k \), where \( A^{(k)} \) has \( \lambda_k \) orbits.
If \textit{lab} is true, \( G^\beta \) is a canonical labelling of \( G \) when the algorithm terminates.

Proof: Apart from minor complications, the theorem follows from the results in [3] and [4].

A simple method for generating \( \text{Aut}(G) \) from \( \Sigma^{(0)} \) is given in [4], as are a few other facts about \( \Sigma^{(0)} \), for example the following lemma.

\textbf{Lemma 2.} Suppose that for some \( W \subseteq V \), \( A_W \) has exactly one non-trivial orbit. Then \( \Sigma^{(0)} \) has a subset which generates a conjugate of \( A_W \) in \( A \).

5. \textbf{EXPERIMENTAL PERFORMANCE}

The algorithm has been implemented in (partly non-standard) Fortran on a CDC Cyber 70 Model 73 computer. The graph \( G \) is represented by its adjacency matrix, stored one bit per entry. The storage of the partitions \( \pi_1 \) is facilitated by the easily verified fact that in any sequence of partitions \( \pi_1 \geq \pi_2 \geq \cdots \), the total number of different cells is less than \( 2n \). Suppose that \( n \) bits occupy \( m \) machine words, and that \( k \) is the maximum value of \( k \) for which \( \pi_k \) is ever computed (obviously, \( k \leq n \)). Then at most \( n(2m + 8) + 2lm \) words of storage are required by the program, plus an extra \( n(m + 2) \) words if \( \text{canon} \ (G) \) is required, and an optional \( 2m(n - 1) \) words to ensure that \( (\Phi(g), \Omega(g)) \) can always be stored at step (8). The function \( \lambda(G, \pi) \) used in the implementation has an integer value formed from the cell-sizes of \( \pi \) and from various items remaining from the computation of \( \pi \) by Algorithm 1.

The execution times for various common families of graphs are shown in Figure 1. For all cases except for the random graphs, the times are for computing \( \text{canon} \ (G) \) as well as \( \text{Aut}(G) \). We believe that both the execution times and their rate of increase with \( n \) are considerably superior to that of any previously published algorithm.

<table>
<thead>
<tr>
<th>Graph Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K_n )</td>
<td>empty graph.</td>
</tr>
<tr>
<td>RD</td>
<td>randomly selected digraph with constant out-degree = ( \frac{1}{2}n ).</td>
</tr>
<tr>
<td>RC</td>
<td>randomly selected circulant graph with degree = ( \frac{1}{2}n ).</td>
</tr>
<tr>
<td>Q_m</td>
<td>( m )-dimensional cube; ( n = 2^m ).</td>
</tr>
<tr>
<td>RG</td>
<td>randomly selected graph with edge-density = ( \frac{1}{2} )</td>
</tr>
<tr>
<td>SR25</td>
<td>strongly regular graphs on 25 vertices (average time).</td>
</tr>
<tr>
<td>SR35</td>
<td>strongly regular block intersection graphs of Steiner triple systems with 15 points and 35 blocks (average time).</td>
</tr>
</tbody>
</table>

The dashed line marked P in Figure 1 gives the time required to perform a single permutation of an adjacency matrix with edge-density \( \frac{1}{2} \). Since this is an essential
step in any program which computes \( \text{canon}(G) \) using an adjacency matrix representation, it can be seen that the algorithm is close to optimal for large random graphs. If \( d(v, w) \) can be computed in time proportional to \(|n|\), it can be shown that the algorithm requires time of at worst order \( n^4 \), provided that \( \xi(\pi) = \theta(\pi) \) for any partition \( \pi \). However, no useful upper bound has been proved in general.

A listing of the program, plus suggestions for implementation, can be obtained from the author.

6. EXAMPLE

Let \( G \) be the graph \( C_5 \times C_5 \) labelled as shown in Figure 2.

The generators of \( \text{Aut}(G) \) found by the algorithm were

\[
(6 \ 21)(7 \ 22)(8 \ 23)(9 \ 24)(10 \ 25)(11 \ 16)(12 \ 17)(13 \ 18)(14 \ 19)(15 \ 20),
(2 \ 5)(3 \ 4)(7 \ 10)(8 \ 9)(12 \ 15)(13 \ 14)(17 \ 20)(18 \ 19)(22 \ 25)(23 \ 24),
(2 \ 6)(3 \ 11)(4 \ 16)(5 \ 21)(8 \ 12)(9 \ 17)(10 \ 22)(14 \ 18)(15 \ 23)(20 \ 24) \quad \text{and}
(1 \ 2)(3 \ 5)(6 \ 7)(8 \ 10)(11 \ 12)(13 \ 15)(16 \ 17)(18 \ 20)(21 \ 22)(23 \ 25),
\]
of which the first generates the stabiliser of \{1, 2\} and the first three generate the stabiliser of 1. \(\text{Aut}(G)\) is transitive and has order 200. The time taken was 0.16 seconds.

![Graph Image]

**Figure 2**

REFERENCES


