PRACTICAL GRAPH ISOMORPHISM

Brendan D. McKay
Computer Science Department
Vanderbilt University
Nashville, Tennessee 37235

Current address:
Department of Computer Science
Australian National University
Canberra, ACT 0200, Australia
bdm@cs.anu.edu.au

We develop an improved algorithm for canonically labelling a graph and finding generators for its automorphism group. The emphasis is on the power of the algorithm for solving practical problems, rather than on the theoretical niceties of the algorithm. The result is an implementation which can successfully handle many graphs with a thousand or more vertices, and is very likely the most powerful graph isomorphism program currently in use.

INTRODUCTION

In this paper we discuss the design of an algorithm for canonically labelling a vertex-coloured graph and for finding generators for its automorphism group. This algorithm is a descendant of one described in McKay [14], which in turn was descended from one which first appeared in McKay [12]. Other algorithms which also employ some of the ideas used by our algorithm include those of Mathon [11], Arlazarov, Zuev, Uskov and Faradzev [1] and Beyer and Proskurowski [2]. However, we are confident that our algorithm is significantly more powerful than any other published algorithm for the practical solution of the isomorphism problem for general graphs. On the few occasions where the proof of a non-trivial assertion is not given here, it can be found in McKay [15].

1.1 Sets and graphs

In this paper V will always denote the set $\{1,2,\cdots,n\}$. The set of all labelled simple graphs with vertex set V will be denoted by $\mathcal{G}(V)$. If $G \in \mathcal{G}(V)$ and $v \in V$, $N(v,G)$ is the set of all elements of V which are adjacent to v in G. Any other graph theoretic concepts not defined here can be found in [4].

Let X be a set and let $\leq$ be a linear (total) order on X. Suppose Z is a set whose elements are finite sequences of elements of

Congressus Numerantium, Vol. 30 (1981), pp. 45- 87

X (the length may vary). Then the *lexicographic ordering of Z induced by* $\leq$ is the linear order $\leq$ defined as follows. If $\alpha = (x_1, x_2, \cdots, x_k) \in Z$ and $\beta = (y_1, y_2, \cdots, y_l) \in Z$ then $\alpha \leq \beta$ either of the following are true.

(i) For some t, $1 \leq t \leq \min\{k, l\}$, we have $x_i = y_i$ for $i < t$ and $x_t < y_t$.

(ii) $x_i = y_i$ for $1 \leq i \leq k$ and $l \geq k$.

If X is a linearly ordered set, then $\min X$ denotes the minimum element of X. In particular, $\min \emptyset = \infty$. The function max is defined similarly.

1-2 Partitions

A *partition* of the set V is a set of disjoint non-empty subsets of V whose union is V. An *ordered partition* of V is a sequence $(V_1, V_2, \cdots, V_r)$, such that $\{V_1, V_2, \cdots, V_r\}$ is a partition of V. The set of all partitions of V and the set of all ordered partitions of V will be denoted by $\Pi(V)$ and $\underline{\Pi}(V)$ respectively. For notational economy we also define $\Pi^*(V) = \Pi(V) \cup \underline{\Pi}(V)$.

The elements of a partition (or ordered partition) $\pi \in \Pi^*(V)$ are usually called its *cells*. A *trivial* cell of π is a cell of cardinality one; the element of such a cell is said to be *fixed* by π. If every cell of π is trivial, then π is a *discrete* partition, while if there is only one cell, π is the *unit* partition.

If π_1, $\pi_2 \in \Pi^*(V)$, we write $\pi_1 \simeq \pi_2$ if π_1 and π_2 have the same cells, in some order. We say that π_1 is *finer* than π_2, denoted $\pi_1 \leq \pi_2$, if every cell of π_1 is a subset of some cell of π_2. Under the same conditions, π_2 is *coarser* than π_1. It is well known that the set $\Pi(V)$ forms a lattice under the partial order $\leq$. This means that, given π_1, $\pi_2 \in \Pi(V)$, there is a unique coarsest partition $\pi_1 \wedge \pi_2 \in \Pi(V)$ such that $\pi_1 \geq \pi_1 \wedge \pi_2$ and $\pi_2 \geq \pi_1 \wedge \pi_2$, and a unique finest partition $\pi_1 \vee \pi_2 \in \Pi(V)$ such that $\pi_1 \leq \pi_1 \vee \pi_2$ and $\pi_2 \leq \pi_1 \vee \pi_2$. Each cell of $\pi_1 \wedge \pi_2$ is a non-empty intersection of a cell of π_1 and a cell of π_2. Each cell of $\pi_1 \vee \pi_2$ is a minimal non-empty subset of V which is both a union of cells of π_1 and a union of cells of π_2.

Let $\pi \in \Pi^*(V)$. Then $\mathrm{fix}(\pi)$ is the set of elements of V which are fixed by π. The *support* of π is the set $\mathrm{supp}(\pi) = V \setminus \mathrm{fix}(\pi)$. The set of *minimum cell representatives* of π is $\mathrm{mcr}(\pi) = \{\min V_i \mid V_i \in \pi\}$, where the minima are under the natural ordering of V.

1·3 **Lemma** *Let* $\pi_1, \pi_2 \in \Pi^*(V)$.

(a) $\mathrm{fix}(\pi_1 \vee \pi_2) = \mathrm{fix}(\pi_1) \cap \mathrm{fix}(\pi_2)$

(b) $\mathrm{fix}(\pi_1 \wedge \pi_2) \supseteq \mathrm{fix}(\pi_1) \cup \mathrm{fix}(\pi_2)$

(c) $\mathrm{supp}(\pi_1 \vee \pi_2) = \mathrm{supp}(\pi_1) \cup \mathrm{supp}(\pi_2)$

(d) $\mathrm{supp}(\pi_1 \wedge \pi_2) \subseteq \mathrm{supp}(\pi_1) \cap \mathrm{supp}(\pi_2)$

(e) $\mathrm{mcr}(\pi_1 \vee \pi_2) \subseteq \mathrm{mcr}(\pi_1) \cap \mathrm{mcr}(\pi_2)$

(f) $\mathrm{mcr}(\pi_1 \wedge \pi_2) = \mathrm{mcr}(\pi_1) \cup \mathrm{mcr}(\pi_2)$ □

Let $\pi = (V_1, V_2, \cdots, V_r) \in \underline{\Pi}(V)$. For each $x \in V$ define $u(x, \pi) = i$, where $x \in V_i$. If $\pi_1, \pi_2 \in \underline{\Pi}(V)$ then we say that π_1 and π_2 are *consistent* if, for any $x, y \in V$, $u(x, \pi_1) < u(y, \pi_1)$ implies that $u(x, \pi_2) \leq u(y, \pi_2)$. As a relation, consistency is symmetric but not transitive. If $\pi_1 \leq \pi_2$ and π_1 and π_2 are consistent, we indicate this by writing $\pi_1 \preceq \pi_2$ or $\pi_2 \succeq \pi_1$. The relation $\preceq$ is transitive but not symmetric.

1·4 Groups

For permutation group theory not delineated here see Wielandt [19]. Let γ be a permutation on V (in other words $\gamma \in S_n$). The image of $v \in V$ under γ will be denoted by v^γ. More generally, if $W \subseteq V$ then $W^\gamma = \{w^\gamma \mid w \in W\}$. Similarly, if $\pi = (V_1, V_2, \cdots, V_r) \in \underline{\Pi}(V)$, then $\pi^\gamma = (V_1^\gamma, V_2^\gamma, \cdots, V_r^\gamma)$. Finally, if $G \in \underline{G}(V)$ then $G^\gamma \in \underline{G}(V)$ has $E(G^\gamma) = \{x^\gamma y^\gamma \mid xy \in E(G)\}$.

If $\Omega \subseteq S_n$, then Ω defines a partition $\theta(\Omega) \in \Pi(V)$ whose cells are the orbits of $\langle \Omega \rangle$, the group generated by Ω. For notational convenience we will write $\theta(\{\gamma\})$ as $\theta(\gamma)$, and $\mathrm{fix}(\Omega)$, $\mathrm{supp}(\Omega)$ and $\mathrm{mcr}(\Omega)$ will be used as abbreviations for $\mathrm{fix}(\theta(\Omega))$, $\mathrm{supp}(\theta(\Omega))$ and $\mathrm{mcr}(\theta(\Omega))$, respectively. The next lemma follows easily from Lemma 1·3.

1·5 **Lemma** *Let $\Omega, \Phi \subseteq S_n$. Then*

(a) $\theta(\Omega \cup \Phi) = \theta(\Omega) \vee \theta(\Phi)$

(b) $\mathrm{fix}(\Omega \cup \Phi) = \mathrm{fix}(\Omega) \cap \mathrm{fix}(\Phi)$

(c) $\mathrm{supp}(\Omega \cup \Phi) = \mathrm{supp}(\Omega) \cup \mathrm{supp}(\Phi)$, *and*

(d) $\mathrm{mcr}(\Omega \cup \Phi) \subseteq \mathrm{mcr}(\Omega) \cap \mathrm{mcr}(\Phi)$. □

Let $\Gamma \leq S_n$ and let Ω be any set such that an action of each $\gamma \in \Gamma$ is defined on each element of Ω. Then the *stabiliser* of Ω in Γ is the group $\Gamma_\Omega = \{\gamma \in \Gamma \mid \omega^\gamma = \omega \text{ for each } \omega \in \Omega\}$. Elements of Γ_Ω are said to *fix* Ω. The most important cases of this construction are as follows.

(i) (*point-wise stabiliser*)

If $W \subseteq V$ then $\Gamma_W = \{\gamma \in \Gamma \mid x^\gamma = x \text{ for each } x \in W\}$.

If $W = \{x_1, x_2, \cdots, x_r\}$ we will also write Γ_W as $\Gamma_{x_1, x_2, \cdots, x_r}$.

(ii) (*set-wise stabiliser*)

If $W \subseteq V$ then $\Gamma_{\{W\}} = \{\gamma \in \Gamma \mid W_\gamma = W\}$.

(iii) (*partition stabiliser*)

If $\pi \in \Pi^*(V)$ has cells $V_1, V_2, \cdots, V_r$ then $\Gamma_\pi = \{\gamma \in \Gamma \mid V_i^\gamma = V_i \text{ for } 1 \leq i \leq r\}$. Note that this is quite different from $\Gamma_{\{\pi\}} = \{\gamma \in \Gamma \mid \pi^\gamma = \pi\}$, unless $\pi \in \underline{\Pi}(V)$.

(iv) (*automorphism group of graph*)

If $G \in \underline{G}(V)$, then the *automorphism group* of G is the group $\mathrm{Aut}(G) = (S_n)_{\{G\}} = \{\gamma \in S_n \mid G^\gamma = G\}$.

DEVELOPMENT OF THE ALGORITHM

In this section we describe the theoretical basis for the algorithm. The more mundane aspects of its implementation will be treated in Section 3.

2·1 Canonical Labels

A *canonical label* is a map $C : \underline{\mathcal{G}}(V) \times \underline{\Pi}(V) \to \underline{\mathcal{G}}(V)$, such that for any $G \in \underline{\mathcal{G}}(V), \pi \in \underline{\Pi}(V)$ and $\gamma \in S_n$ we have

(C1) $C(G,\pi) \cong G$

(C2) $C(G^\gamma, \pi^\gamma) = C(G,\pi)$

(C3) If $C(G,\pi^\gamma) = C(G,\pi)$, then $\pi^\gamma = \pi^\delta$ for some $\delta \in \mathrm{Aut}(G)$.

The main use of a canonical label is to solve various graph isomorphism problems as indicated in the following theorem.

2·2 **Theorem** *Let G_1, $G_2 \in \underline{\mathcal{G}}(V)$, $\pi \in \underline{\Pi}(V)$ and $\gamma \in S_n$. Then $C(G_1,\pi) = C(G_2,\pi^\gamma)$ if and only if there is a permutation $\delta \in S_n$ such that $G_2 = G_1^\delta$ and $\pi^\gamma = \pi^\delta$.*

Proof: The existence of δ as required implies that $C(G_1,\pi) = C(G_2,\pi^\gamma)$ by Property C2. Suppose conversely that $C(G_1,\pi) = C(G_2,\pi^\gamma)$. By Property C1, $G_2 = G_1^\beta$ for some $\beta \in S_n$. Therefore $C(G_2,\pi^\gamma) = C(G_1^\beta,\pi^\gamma) = C(G_1,\pi^{\gamma\beta^{-1}})$, by Property C2. Since $C(G_1,\pi) = C(G_2,\pi^\gamma)$, there is some $\alpha \in \mathrm{Aut}(G_1)$ such that $\pi^{\gamma\beta^{-1}} = \pi^\alpha$, by Property C3, and so $\pi^\gamma = \pi^{\alpha\beta}$. But $\alpha \in \mathrm{Aut}(G_1)$, and so $G_2 = G_1^\beta = G_1^{\alpha\beta}$. □

The isomorphism problem described in Theorem 2·2 can be thought of as that of testing vertex-coloured graphs for isomorphism. Given $|\pi|$ colours, we colour the vertices of G_1 which lie in the i-th cell of π with the i-th colour, for $1 \le i \le |\pi|$. We then similarly colour the vertices of G_2 in accordance with π^γ. This will use the same colours with the same frequency. Theorem 2·2 now says that $C(G_1,\pi) = C(G_2,\pi^\gamma)$ if and only if there is a colour-preserving isomorphism from G_1 to G_2.

The most important case is, of course, when π is the unit partition (V), in which case Property C3 holds trivially. However we will maintain the more general setting we have created, since the added complications will only be slight.

2·3 Equitable Partitions

For $G \in \underline{G}(V)$, $v \in V$ and $W \subseteq V$, we define $d_G(v, W)$ to be the number of elements of W which are adjacent in G to v. The subscript G will normally be suppressed. We will say that $\pi \in \Pi^*(V)$ is *equitable* (with respect to G) if, for all $V_1, V_2 \in \pi$ (not necessarily distinct) we have $d(v_1, V_2) = d(v_2, V_2)$. It is easy to show that the equitable members of $\Pi(V)$ form a lattice which is closed under $\vee$. Since the discrete partition is always equitable, it follows that for every $\pi \in \Pi(V)$ there is a unique coarsest equitable partition $\xi(\pi) \in \Pi(V)$ which is finer than π.

One of our first concerns in this section will be to study an efficient procedure for computing $\xi(\pi)$ from π.

2·4 The Refinement Procedure

The algorithm we give here is a descendant of one first described in McKay [12]. It actually turns out to be a generalization of an algorithm of Hopcroft ([8], see also [7]) for minimizing the number of states in a finite automaton, although it was not derived from the latter.

The algorithm accepts a graph $G \in \underline{G}(V)$, an ordered partition $\pi \in \underline{\Pi}(V)$ and a sequence $\alpha = (W_1, W_2, \cdots, W_M)$ of distinct cells of π. The result is an ordered partition $\mathcal{R}(G, \pi, \alpha) \in \underline{\Pi}(V)$. Under suitable conditions, to be discussed below, $\mathcal{R}(G, \pi, \alpha) \simeq \xi(\pi)$.

2·5 Algorithm *Compute* $\mathcal{R}(G, \pi, \alpha)$ *given* $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ *and* $\alpha = (W_1, W_2, \cdots, W_M) \subseteq \pi$.

(1) $\tilde{\pi} := \pi$

$m := 1$

(2) If ($\tilde{\pi}$ is discrete or $m > M$) *stop*: $\mathcal{R}(G, \pi, \alpha) = \tilde{\pi}$

$W := W_m$

$m := m + 1$

$k := 1$

{Suppose $\tilde{\pi} = (V_1, V_2, \cdots, V_r)$ at this point.}

(3) Define $(X_1, X_2, \cdots, X_s) \in \underline{\Pi}(V_k)$ such that for any $x \in X_i$, $y \in X_j$ we have $d(x, W) < d(y, W)$ if and only if $i < j$.

If $(s = 1)$ go to (4)

Let t be the smallest integer such that $|X_t|$ is maximum $(1 \leq t \leq s)$.

If $(W_j = V_k$ for some j $(m \leq j \leq M))$ $W_j := X_t$

For $1 \leq i < t$ set $W_{M+i} := X_i$

For $t < i \leq s$ set $W_{M+i-1} := X_i$

$M := M + s - 1$

Update $\tilde{\pi}$ by replacing the cell V_k with the cells $X_1, X_2, \cdots, X_s$ in that order (*in situ*).

(4) $k := k + 1$

If $(k \leq r)$ go to (3)

Go to (2) □

2-6 **Theorem** *For any $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$, we have $\mathcal{R}(G, \pi, \pi) = \xi(\pi)$.*

Proof: (a) The value of $M - m$ is decreased in Step (2) and is only increased when $\tilde{\pi}$ is made strictly finer. Therefore the algorithm is certain to terminate.

(b) By definition, $\xi(\pi) \leq \pi$, so $\xi(\pi) \leq \tilde{\pi}$ at Step (1). Now suppose that $\xi(\pi) \leq \tilde{\pi}$ before some execution of Step (3). Since W is a cell of some partition coarser than $\xi(\pi)$ (ie. some earlier value of $\tilde{\pi}$), it is a union of cells of $\xi(\pi)$. Since $\xi(\pi)$ is equitable, we must have that $\xi(\pi) \leq \tilde{\pi}$ after the execution of Step (3). Therefore, by induction, $\xi(\pi) \leq \mathcal{R}(G, \pi, \pi) \leq \pi$ when the algorithm stops.

(c) Suppose that $\mathcal{R}(G, \pi, \pi)$ is not equitable. Then for some Y_1, $Y_2 \in \mathcal{R}(G, \pi, \pi)$ there are x, $y \in Y_1$ such that $d(x, Y_2) \neq d(y, Y_2)$. Since $\tilde{\pi}$

is made successively finer by the algorithm, x and y must always be in the same cell of $\tilde{\pi}$.

(d) At Step (1), Y_2 is contained in some element of α. Hence Y_2 must sometime be contained in W for an execution of Step (3).

(e) Since x and y are never separated, $d(x,W) = d(y,W)$. But since W is a union of cells of $\mathcal{R}(G,\pi,\pi)$, and $d(x,Y_2) \neq d(y,Y_2)$, there is at least one other cell Y_3 of $\mathcal{R}(G,\pi,\pi)$ contained in W for which $d(x,Y_3) \neq d(y,Y_3)$. Since Y_2 and Y_3 are different cells of $\mathcal{R}(G,\pi,\pi)$ they must be separated at some execution of Step (3). At least one of them, say Y_2 will then be contained in some new element of α.

(f) Since the argument in (e) can clearly be repeated indefinitely, the algorithm never stops, contradicting (a). Therefore our assumption that $\mathcal{R}(G,\pi,\pi)$ is not equitable must be false, which proves that $\mathcal{R}(G,\pi,\pi) \simeq \xi(\pi)$. □

An important advantage that Algorithm 2·5 has over previous algorithms for computing $\xi(\pi)$ is that α can sometimes be chosen to be a proper subset of π. One method of choosing α is described in the next theorem.

2·7 **Theorem** *Let $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ and suppose that there is some equitable partition π' which is coarser than π. Choose $\alpha \subseteq \pi$ such that for any $W \in \pi'$, we have $X \subseteq W$ for at most one $X \in \pi \setminus \alpha$. Then $\mathcal{R}(G,\pi,\alpha) \simeq \xi(\pi)$.*

Proof: (a) By the same arguments as in Theorem 2·6, the algorithm will eventually stop, and $\xi(\pi) \leq \mathcal{R}(G,\pi,\alpha) \leq \pi$.

(b) Suppose that $\mathcal{R}(G,\pi,\alpha)$ is not equitable. Then for some Y_1, $Y_2 \in \mathcal{R}(G,\pi,\alpha)$ there are x, $y \in Y_1$ such that $d(x,Y_2) \neq d(y,Y_2)$. Since $\mathcal{R}(G,\pi,\alpha) \leq \pi'$, and π' is equitable, there is at least one other cell Y_3 of $\mathcal{R}(G,\pi,\alpha)$ such that $d(x,Y_3) \neq d(y,Y_3)$.

(c) If Y_2 and Y_3 are in different cells of π, the defined relationship between π, α and π' ensures that at least one of them, say Y_2, is contained in some cell of α at Step (1). We can then take up the proof of Theorem 2·6 at step (d), and conclude that $\mathcal{R}(G,\pi,\alpha) \simeq \xi(\pi)$.

(d) On the other hand, Y_2 and Y_3 may be in the same cell of π. Since they are in different cells of $\mathcal{R}(G,\pi,\alpha)$ they must be separated at Step (3). At least one of them, say Y_2, will then be contained in some new element of α. We can now take up the proof of Theorem 2·6 at step (e) and conclude as before that $\mathcal{R}(G,\pi,\alpha) \simeq \xi(\pi)$. □

One application of Theorem 2·7 occurs when G is regular and π has more than one cell. The unit partition π_0 is equitable, and so we can choose α to be π less any one cell. This will be particularly time-saving if $\pi = (v, V \setminus v)$ for some v, in which case we can use $\alpha = (v)$.

A much more important application of Theorem 2·7 will be described in Section 2·9.

Two very useful properties of Algorithm 2·5 are stated in the next lemma. Both of them are immediate consequences of the definition of the algorithm.

2·8 **Lemma** *Let $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$, α an ordered subset of π and $\gamma \in S_n$. Then*

(a) $\mathcal{R}(G,\pi,\alpha) \preceq \pi$, *and*

(b) $\mathcal{R}(G^\gamma,\pi^\gamma,\alpha^\gamma) = \mathcal{R}(G,\pi,\alpha)^\gamma$. □

2·9 Partition nests

Let $\pi = (v_1, v_2, \cdots, v_k) \in \underline{\Pi}(V)$ and let $v \in V_i$ for some i. If $|V_i| = 1$ define $\pi \circ v = \pi$. If $|V_i| > 1$ define $\pi \circ v = (V_1, \cdots, V_{i-1}, v, V_i \setminus v, V_{i+1}, \cdots, V_k)$. Also define $\pi \perp v = \mathcal{R}(G, \pi \circ v, (v))$.

Given $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ and a sequence $\mathbf{v} = v_1, v_2, \cdots, v_{m-1}$ of distinct elements of V, we define the *partition nest derived from G, π and $\mathbf{v}$* to be $[\pi_1, \pi_2, \cdots, \pi_m]$, where

(a) $\pi_1 = \mathcal{R}(G,\pi,\pi)$, and

(b) $\pi_i = \pi_{i-1} \perp v_{i-1}$, for $2 \leq i \leq m$.

It follows from Theorems 2·6 and 2·7 that each π_i is equitable. Define $\underline{N}(V)$ to be the set of all partition nests derived from some $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ and vector $\mathbf{v}$ of distinct elements of V.

2·10 The basic search tree

Let $G \in \underline{G}(V)$ and $\pi \in \underline{\Pi}(V)$. Then the *search tree* $T(G,\pi)$ is the set of all partition nests $\nu = [\pi_1, \pi_2, \cdots, \pi_m] \in \underline{N}(V)$ such that ν is derived from G, π and a sequence $v_1, v_2, \cdots, v_{m-1}$ where, for $1 \leq i \leq m-1$, v_i is an element of the first non-trivial cell of π_i which has the smallest size. This definition implies that $|\pi_i| < |\pi_{i+1}|$ for $1 \leq i < m$.

The elements of $T(G,\pi)$ will be referred to as *nodes*. The *length* $|\nu|$ of a node ν is the number of partitions it contains. If $\nu = [\pi_1, \pi_2, \cdots, \pi_m]$ is a node then $\nu^{(i)}$ denotes the node $[\pi_1, \pi_2, \cdots, \pi_i]$, for $1 \leq i \leq m$. Thus $\nu^{(m)} = \nu$. If $m \geq 2$ then ν is called a *successor* of $\nu^{(m-1)}$. Similarly, ν is a *descendant* of $\nu^{(i)}$ (and $\nu^{(i)}$ is an *ancestor* of ν) if $1 \leq i < m$. The *root node* $[\pi_1]$ is an ancestor of every node other than itself. The set of all nodes equal to or descended from a node ν constitutes the *subtree* of $T(G,\pi)$ *rooted* at ν, and is denoted by $T(G,\pi,\nu)$. If the last partition in a node is discrete, ν will be called a *terminal node*.

Suppose that ν_1 and ν_2 are distinct nodes, neither of which is a descendant of the other. Then for some i, $\nu_1^{(i)} = \nu_2^{(i)}$ but $\nu_1^{(i+1)} \neq \nu_2^{(i+1)}$. The node $\nu_1^{(i+1)}$ will be denoted by $\nu_1 - \nu_2$ and $\nu_2^{(i+1)}$ by $\nu_2 - \nu_1$.

The natural linear ordering of V can be used to provide an ordering $<$ of the nodes of $T(G,\pi)$. Let ν_1 and ν_2 be distinct nodes. If ν_1 is an ancestor of ν_2 then $\nu_1 < \nu_2$. If neither of ν_1 or ν_2 is an ancestor of the other, there is a node $[\pi_1, \pi_2, \cdots, \pi_m]$ and vertices $v_1 \neq v_2$ such that $\nu_1 - \nu_2 = [\pi_1, \pi_2, \cdots, \pi_m, \pi_m \perp v_1]$ and $\nu_2 - \nu_1 = [\pi_1, \pi_2, \cdots, \pi_m, \pi_m \perp v_2]$. Then we have $\nu_1 < \nu_2$ if $v_1 < v_2$. If $\nu_1 < \nu_2$, we say that ν_1 is *earlier* than ν_2, and that ν_2 is *later* than ν_1.

Some of the obvious properties of this ordering of $T(G,\pi)$ are listed in the next lemma.

2·11 Lemma *Let $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ and $\nu_1, \nu_2, \nu_3 \in T(G,\pi)$. Then*

(a) *Exactly one of $\nu_1 < \nu_2$, $\nu_1 = \nu_2$ and $\nu_2 < \nu_1$ is true.*

(b) *If $\nu_1 < \nu_2$ and $\nu_2 < \nu_3$ then $\nu_1 < \nu_3$.*

(c) *If* $\nu_1 < \nu_2$, $\nu_1' \in T(G,\pi,\nu_1)$ *and* $\nu_2' \in T(G,\pi,\nu_2)$ *then* $\nu_1' < \nu_2'$, *except possibly if* ν_1 *is an ancestor of* ν_2.

(d) *If* $\nu_1 \neq \nu_2$ *and neither of* ν_1 *and* ν_2 *is an ancestor of the other, then* $\nu_1 < \nu_2$ *if and only if* $\nu_1 - \nu_2 < \nu_2 - \nu_1$. □

Given $G \in \underline{G}(V)$ and $\pi \in \underline{\Pi}(V)$ we can generate the elements of $T(G,\pi)$ in the order given by $<$, with the simple backtrack algorithm given below.

2·12 **Algorithm** *Generate* $T(G,\pi)$ *in the order earliest to latest, given* $G \in \underline{G}(V)$ *and* $\pi \in \underline{\Pi}(V)$.

(1) $k := 1$

$\pi_1 := \mathcal{R}(G,\pi,\pi)$

Output $[\pi_1]$

(2) If (π_k is discrete) go to (4)

$W_k :=$ first non-trivial cell of π_k of the smallest size.

(3) If ($W_k = \emptyset$) go to (4)

$v := \min W_k$

$W_k := W_k \setminus v$

$\pi_{k+1} := \pi_k \perp v$

$k := k + 1$

Output $[\pi_1, \pi_2, \cdots, \pi_k]$

Go to (2)

(4) $k := k - 1$

If ($k \geq 1$) go to (3)

Stop: All the nodes of $T(G,\pi)$ have been output in the required order. □

2·13 Group Actions on $T(G,\pi)$

If $\nu = [\pi_1, \pi_2, \cdots, \pi_m] \in \underline{N}(V)$ and $\gamma \in S_n$, then we can define $\nu^\gamma = [\pi_1^\gamma, \pi_2^\gamma, \cdots, \pi_m^\gamma]$. Obviously $\nu^\gamma \in \underline{N}(V)$. The property of Algorithm 2·5 described in Lemma 2·8 has immediate consequences for $T(G,\pi)$, as we describe in the next theorem.

2·14 Theorem *Let $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ and $\gamma \in S_n$.*

(a) $T(G^\gamma, \pi^\gamma) = T(G,\pi)^\gamma$.

(b) *If $\nu \in T(G,\pi)$, then* $T(G^\gamma, \pi^\gamma, \nu^\gamma) = T(G,\pi,\nu)^\gamma$. □

The map from $T(G,\pi)$ to $T(G,\pi)^\gamma$ induced by γ will not in general preserve the ordering $<$.

We will be particularly interested in permutations $\gamma \in S_n$ such that $G^\gamma = G$ and $\pi^\gamma \in \pi$. In other words, $\gamma \in \mathrm{Aut}(G)_\pi$. If ν_1, $\nu_2 \in T(G,\pi)$ and $\nu_2 = \nu_1^\gamma$ for some $\gamma \in \mathrm{Aut}(G)_\pi$ we write $\nu_1 \sim \nu_2$ and say that ν_1 and ν_2 are *equivalent.* By Theorem 2·14, $\sim$ is an equivalence relation on $T(G,\pi)$. If ν is a terminal node of $T(G,\pi)$ then ν is called an *identity node* if there is no earlier node of $T(G,\pi)$ which is equivalent to ν.

The following theorem is fundamental to our treatment of group actions on $T(G,\pi)$.

2·15 Theorem *Let $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ and $\gamma \in \mathrm{Aut}(G)_\pi$. Then*

(a) $T(G,\pi)^\gamma = T(G,\pi)$.

(b) *If $\nu \in T(G,\pi)$, then* $T(G,\pi,\nu^\gamma) = T(G,\pi,\nu)^\gamma$.

(c) *If ν_1, $\nu_2 \in T(G,\pi)$, $\nu_1 < \nu_2$ and $\nu_1 \sim \nu_2$, then $T(G,\pi,\nu_2 - \nu_1)$ contains no identity nodes.*

Proof: Assertions (a) and (b) are immediate consequences of Theorem 2·14, so we consider only assertion (c). If $\nu_1 \sim \nu_2$, there is some $\gamma \in \mathrm{Aut}(G)_\pi$ such that $\nu_2 = \nu_1^\gamma$. But then $\nu_2 - \nu_1 = (\nu_1 - \nu_2)^\gamma$ and so $T(G,\pi,\nu_2 - \nu_1) = T(G,\pi,\nu_1 - \nu_2)^\gamma$ by (b). However $\nu_1 < \nu_2$ and so $\nu_1 - \nu_2 < \nu_2 - \nu_1$, by Lemma 2·11. Therefore, every terminal node in $T(G,\pi,\nu_2 - \nu_1)$ is equivalent to an earlier terminal node in $T(G,\pi,\nu_1 - \nu_2)$, which proves (c).

2·16 **Indicator functions**

Let Δ be any linearly ordered set. An *indicator function* is a map $\Lambda : \underline{\mathcal{G}}(V) \times \underline{\Pi}(V) \times \underline{N}(V) \to \Delta$ such that $\Lambda(G^\gamma, \pi^\gamma, \nu^\gamma) = \Lambda(G, \pi, \nu)$ for any $G \in \underline{\mathcal{G}}(V)$, $\pi \in \underline{\Pi}(V)$, $\nu \in T(G, \pi)$ and $\gamma \in S_n$.

Given one indicator function Λ, we can define another indicator function $\underline{\Lambda}$ by

$$\underline{\Lambda}(G, \pi, \nu) = (\Lambda(G, \pi, \nu^{(1)}), \Lambda(G, \pi, \nu^{(2)}), \cdots, \Lambda(G, \pi, \nu^{(k)})),$$

where $k = |\nu|$, with the lexicographic ordering induced from the ordering of Δ.

2·17 **Definition of $C(G, \pi)$**

If $\nu = [\pi_1, \pi_2, \cdots, \pi_m]$ is a terminal node of $T(G, \pi)$ then π_m is a discrete ordered partition, by definition. This means that π_m defines an ordering of the elements of V. We can define a graph $G(\nu)$ isomorphic to G by relabelling the vertices of G in the order that they appear in π_m. More precisely, if $\pi_m = (v_1 \mid v_2 \mid \cdots \mid v_n)$, and $\delta \in S_n$ is the permutation taking v_i onto i for $1 \leq i \leq n$, then $G(\nu) = G^\delta$. The following lemma is an immediate consequence of the definitions.

2·18 **Lemma** *If $G \in \underline{\mathcal{G}}(V)$, $\pi \in \underline{\Pi}(V)$, $\gamma \in S_n$ and $\nu \in T(G, \pi)$ is a terminal node, then $G(\nu^\gamma) = G(\nu)$ if and only if $\gamma \in \mathrm{Aut}(G)$.*

Proof: Let $\nu = [\pi_1, \pi_2, \cdots, \pi_m]$, where $\pi_m = (v_1 \mid v_2 \mid \cdots \mid v_n)$, and take the permutation $\delta \in S_n$ which takes v_i onto i for $1 \leq i \leq n$). Then $G(\nu) = G^\delta$ by definition. Also by definition, $\pi_m^\gamma = (v_1^\gamma \mid v_2^\gamma \mid \cdots \mid v_n^\gamma)$, and so $G(\nu^\gamma) = G^{\gamma^{-1}\delta}$. Therefore $G(\nu) = G(\nu^\gamma)$ if and only if $G^\delta = G^{\gamma^{-1}\delta}$, which is possible if and only if $\gamma \in \mathrm{Aut}(G)$. □

Our next requirement is a linear ordering of $\underline{\mathcal{G}}(V)$. Any such ordering will do, but it will be convenient for us to use an ordering defined using the adjacency matrices of elements of $\underline{\mathcal{G}}(V)$. Given $G \in \underline{\mathcal{G}}(V)$ we can define an integer $n(G)$ by writing down the elements of the adjacency matrix in a row-by-row fashion, and interpreting the result as

an n^2-bit binary number. If G_1, $G_2 \in \underline{G}(V)$ we can then define $G_1 \leq G_2$ if and only if $n(G_1) \leq n(G_2)$.

We can at last define $C(G,\pi)$. Let $X(G,\pi)$ be the set of all terminal nodes of $T(G,\pi)$. Choose an arbitrary (but fixed) indicator function Λ. Let $\Lambda^* = \max\{\underline{\Lambda}(G,\pi,\nu) \mid \nu \in X(G,\pi)\}$. Then we define $C(G,\pi) = \max\{G(\nu) \mid \nu \in X(G,\pi)$ and $\underline{\Lambda}(G,\pi,\nu) = \Lambda^*\}$.

2·19 **Theorem** *C is a canonical label.*

Proof: We show that C has Properties C1–C3 (Section 2·1). Property C1 is true because $G(\nu) \cong G$ for any $\nu \in X(G,\pi)$. Now let $\gamma \in S_n$. By Theorem 2·14, $T(G^\gamma,\pi^\gamma) = T(G,\pi)^\gamma$ and so $X(G^\gamma,\pi^\gamma) = X(G,\pi)^\gamma$. Also, by the definition of indicator function, $\underline{\Lambda}(G^\gamma,\pi^\gamma,\nu^\gamma) = \underline{\Lambda}(G,\pi,\nu)$ for any $\nu \in X(G,\pi)$. Finally, by the definition of $G(\nu)$, we find that $G^\gamma(\nu^\gamma) = G(\nu)$. Therefore C has Property C2.

In order to prove Property C3 we must recall Lemma 2·8(a). Together with the fact that any $\nu \in X(G,\pi)$ is a partition nest, this implies that $C(G,\pi) = G^\delta$ for some $\delta \in S_n$ such that $\pi^\delta = \pi$.

Now suppose that $C(G,\pi^\gamma) = C(G,\pi)$ for some $\gamma \in S_n$. Since C satisfies Property C2, $C(G,\pi^\gamma) = C(G^{\gamma^{-1}},\pi)$. Therefore there are α, $\beta \in S_n$ such that $\pi^\alpha = \pi^\beta = \pi$, $C(G,\pi^\gamma) = G^{\gamma^{-1}\alpha}$ and $C(G,\pi) = G^\beta$. The assumption that $C(G,\pi^\gamma) = C(G,\pi)$ thus implies that $G^{\gamma^{-1}\alpha} = G^\beta$ and so $\beta\alpha^{-1}\gamma \in \mathrm{Aut}(G)$. Finally, $\pi^{\beta\alpha^{-1}\gamma} = \pi^\gamma$ since $\pi^\beta = \pi^\alpha = \pi$. Therefore C has Property C3. □

An elementary means of computing $C(G,\pi)$ is now apparent. Using Algorithm 2·12 we can generate every element of $X(G,\pi)$. We can then identify those $\nu \in X(G,\pi)$ for which $\underline{\Lambda}(G,\pi,\nu)$ is maximum and so find $C(G,\pi)$ from its definition. It is not necessary to store all of $X(G,\pi)$ simultaneously; its elements can be processed as they are generated and then discarded. However, this process is not practical for use with a great many graphs because of the size of $X(G,\pi)$. One problem is with graphs having large automorphism groups. Since $\mathrm{Aut}(G)$ acts semi-regularly on $X(G,\pi)$, $|X(G,\pi)|$ must be a multiple of $\mathrm{Aut}(G)$, and so can be impossibly large, even for moderate n. Secondly, there are graphs

for which $|X(G,\pi)|$ is very large, even if $|\mathrm{Aut}(G)|$ is small. We will meet some of these graphs in §3.

The method which we will use to attack these difficulties is a process of pruning $T(G,\pi)$. Let us say that $\nu \in X(G,\pi)$ is a *canonical* node if $C(G,\pi) = G(\nu)$. Obviously, any part of $T(G,\pi)$ can be ignored if the remainder is known to contain a canonical node. Our guiding light is the following theorem, which is already implicit in the foregoing.

2·20 **Theorem** *Let $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$, and $\Lambda^* = \max\{\underline{\Lambda}(G,\pi,\nu) \mid \nu \in X(G,\pi)\}$. Let $X^*(G,\pi)$ be any subset of $X(G,\pi)$ which contains those identity nodes ν for which $\underline{\Lambda}(G,\pi,\nu) = \Lambda^*$. Then $X^*(G,\pi)$ contains a canonical node.* □

In the terms of Theorem 2·20 our aim will be to reduce the size of $X^*(G,\pi)$ as much as possible. We will reduce the number of elements of $X^*(G,\pi)$ which are not identity nodes by searching for automorphisms of G and employing any we find to delete subtrees of $T(G,\pi)$. We will reduce the number of identity nodes in $X^*(G,\pi)$ by using Λ.

2·21 **Using automorphisms to prune $T(G,\pi)$**

The existence of one or more automorphisms of G can be inferred during the generation of $T(G,\pi)$ in at least two different ways.

(1) We may find two terminal nodes ν_1, $\nu_2 \in X(G,\pi)$ such that $G(\nu_1) = G(\nu_2)$.

(2) We can sometimes infer the presence of automorphisms from the structure of an equitable partition.

The first case is the more important and will be treated first. The second case can wait until Section 2·24.

Suppose then that during the generation of $T(G,\pi)$ we encounter a terminal node $\nu_2 \in X(G,\pi)$, compute $G(\nu_2)$, and discover that it is the same as $G(\nu_1)$ for some earlier terminal node ν_1. Since ν_1 and ν_2 are terminal nodes, there is a unique permutation $\gamma \in S_n$ such that $\nu_2 = \nu_1^\gamma$. It then follows from Lemma 2·18 that $\gamma \in \mathrm{Aut}(G)$. We will call γ an *explicit* automorphism.

Once we have found an explicit automorphism there are several ways we can put it to work. These are based on Theorem 2·15. The immediate outcome of Theorem 2·15 is that we may ignore the remainder of the subtree $T(G, \pi, \nu_2 - \nu_1)$. However, we can do better than that. Since $\mathrm{Aut}(G)$ is a group, not only γ but all its powers are in $\mathrm{Aut}(G)$. Moreover, if we have found several automorphisms of G, any permutation which is generated by these is also in $\mathrm{Aut}(G)$. The following scheme for handling this mass of information is not always the best, but has been found to work very well in many circumstances.

Let $\zeta \in X(G, \pi)$ be the earliest terminal node. We will need the following lemma.

2·22 **Lemma** *Let* $\nu_1 < \nu_2 \in X(G, \pi)$. *Then* $|\zeta - \nu_2| \leq |\nu_1 - \nu_2|$.

Proof: If $|\nu_1 - \nu_2| < |\zeta - \nu_2|$, then $\nu_2 \in T(G, \pi, \zeta - \nu_1)$, which contradicts the assumption that $\nu_1 < \nu_2$. □

We next introduce an auxiliary partition $\theta \in \Pi(V)$. We initially set θ equal to the discrete partition of V, and whenever we obtain an explicit automorphism γ, we update $\theta := \theta \vee \theta(\gamma)$. This means, by Lemma 1·13, that θ is at every stage the orbit partition of the group generated by all the explicit automorphisms so far discovered. It also means that $\theta \leq \theta(\mathrm{Aut}(G)_{\pi_m})$, where $[\pi_1, \pi_2, \cdots, \pi_m]$ is any common ancestor of all the terminal nodes we have yet considered. This is because a permutation taking one node to another fixes their common ancestors.

Now consider a node $\nu = [\pi_1, \pi_2, \cdots, \pi_m]$ which is an ancestor of ζ. Because of the definition of ζ, ν is also an ancestor of all the terminal nodes generated so far. Let $W = \{v_1, v_2, \cdots, v_k\}$ be the first non-trivial cell of smallest size of π_m, where $v_1 < v_2 < \cdots < v_k$. Since $\theta \leq \pi_m$, θ induces a partition of W. Now the successors of ν, in the order earliest to latest, are $\nu(v_1), \nu(v_2), \cdots, \nu(v_k)$, where $\nu(v_i) = [\pi_1, \pi_2, \cdots, \pi_m, \pi_m \perp v_i]$. If $v_i < v_j$ are in the same cell of θ, there is some automorphism γ, generated by the explicit automorphisms so far discovered, such that $\nu(v_j) = \nu(v_i)^\gamma$. Therefore we can exclude the subtree $T(G, \pi, \nu(v_j))$ from further examination. There are two ways of doing this. The first is that, as we generate successive subtrees $T(G, \pi, \nu(v_1))$, $T(G, \pi, \nu(v_2))$,

$\cdots$ we only consider those for which $v_i \in \mathrm{mcr}(\theta)$. The second is that, upon discovering an explicit automorphism γ during the generation of $T(G, \pi, \nu(v_i))$, and updating θ, we check to see if it still true that $v_i \in \mathrm{mcr}(\theta)$. If not, we have found proof (namely γ) that $T(G, \pi, \nu(v_i))$ only contains terminal nodes equivalent to those of some subtree we have already examined. Therefore we can return at once to ν and consider $\nu(v_{i+1})$.

The technique just described often allows us to jump all the way back to an ancestor ν of ζ after only generating one terminal node of a subtree rooted at a successor of ν. Unfortunately this is not always possible, for example when a new terminal node is not recognised as being equivalent to an earlier one. It will also be possible (due to the use of Λ – see later) for a whole subtree to be ignored without knowing it to be equivalent to anything else. In order to put our automorphisms to work in such cases we have devised the following scheme.

Firstly, we maintain a store Ψ which contains $(\mathrm{fix}(\gamma), \mathrm{mcr}(\gamma))$ for every explicit automorphism γ so far discovered (or some subset of them). Then, with each non-terminal node $\nu \in T(G, \pi)$ we associate a set $W(\nu) \subseteq V$. The first time (if any) we encounter ν in the search of $T(G, \pi)$, $W(\nu)$ is set equal to the first smallest non-trivial cell of π_m, where $\nu = [\pi_1, \pi_2, \cdots, \pi_m]$. The next time we encounter ν (if any), we redefine $W(\nu) := W(\nu) \cap \mathrm{mcr}(\gamma_1) \cap \mathrm{mcr}(\gamma_2) \cap \cdots \cap \mathrm{mcr}(\gamma_r)$, where $\gamma_1, \gamma_2, \cdots, \gamma_r$ are those previously encountered explicit automorphisms which fix ν. From then on we can ignore subtrees $T(G, \pi, \nu(v))$ for which $v \notin W(\nu)$. This is justified by Lemma 1·5. The reasons for deferring the modification of $W(\nu)$ until the second encounter with ν are (i) that the subtree rooted at the earliest successor of ν has to be examined anyway (since the smallest element of $W(\nu)$ before the modification remains in $W(\nu)$ after the modification) and (ii) that there is often no second encounter with ν (we may find an automorphism allowing us to jump back to an ancestor of ν). The next lemma shows that we can determine whether γ fixes ν by looking at $\mathrm{fix}(\gamma)$.

2·23 **Lemma** *Let γ be an explicit automorphism. Let $\nu = [\pi_1, \pi_2, \cdots, \pi_m] \in T(G, \pi)$ be derived from G, π and $v_1, v_2, \cdots, v_{m-1}$. Then γ fixes ν if and only if $\{v_1, v_2, \cdots, v_{m-1}\} \subseteq \mathrm{fix}(\gamma)$.* □

There is one other circumstance under which we may wish to change $W(\nu)$. If we find two equivalent terminal nodes ν_1, ν_2 where $\nu_2 = \nu_1^\gamma$ and where ν is the longest common ancestor of ν_1 and ν_2, we can set $W(\nu) := W(\nu) \cap \mathrm{mcr}(\gamma)$.

2·24 Implicit Automorphisms

There are occasions when we can infer the presence of one or more automorphisms without generating any of them explicitly. These are based on the following lemma.

2·25 **Lemma** *Let $G \in \mathcal{G}(V)$ and let $\pi \in \Pi(V)$ be equitable with respect to G. If π has m non-trivial cells and either $n \leq |\pi| + 4$, $n = |\pi| + m$ or $n = |\pi| + m + 1$, then $\pi_1 = \theta(\mathrm{Aut}(G)_{\pi_1})$ for any equitable $\pi_1 \leq \pi$.* □

The most commonly occurring case of Lemma 2·25 is when $n = |\pi| + m$, which corresponds to π_1 only having cells of size one or two.

Lemma 2·25 can be put to several uses. The most immediate application is that whenever we encounter a node $\nu = [\pi_1, \pi_2, \cdots, \pi_m]$ for which π_m satisfies the requirements of Lemma 2·25, we can infer that all the terminal nodes descended from ν are equivalent, and so at most one of them is an identity node (the earliest one, if any). A less direct technique is to store the pair $(\mathrm{fix}(\pi_m), \mathrm{mcr}(\pi_m))$ in the list Ψ, along with the along with the similar pairs derived from explicit automorphisms. It can then become useful in pruning later parts of the search tree.

2·26 Eliminating identity nodes

The techniques of the last few sections are generally quite efficient in removing terminal nodes which are not identity nodes. However, there are occasions when the number of identity nodes is unmanageably large. Examples of these will be given in later sections. Some of these can be eliminated by means of an indicator function Λ.

Suppose that during the search of $T(G, \pi)$ we maintain a node variable ρ. When the first terminal node ζ is generated, we initialise

$\rho := \zeta$. Thereafter we update $\rho := \nu$ whenever we find a terminal node ν such that $\underline{\Lambda}(G,\pi,\nu) > \underline{\Lambda}(G,\pi,\rho)$ or $\underline{\Lambda}(G,\pi,\nu) = \underline{\Lambda}(G,\pi,\rho)$ and $G(\nu) > G(\rho)$. The definition of $C(G,\pi)$ ensures that by the time we have finished searching $T(G,\pi)$ we have $G(\rho) = C(G,\pi)$, provided the set of terminal nodes examined includes all the identity nodes. Now suppose that at some instant during our search we have $\rho = [\pi_1, \pi_2, \cdots, \pi_m]$ and encounter a node $\nu = [\pi'_1, \pi'_2, \cdots, \pi'_k]$, not necessarily terminal. Let $r = \min\{m,k\}$. Then, if $\underline{\Lambda}(G,\pi,\nu^{(r)}) < \underline{\Lambda}(G,\pi,\rho^{(r)})$, the definition of an indicator function tells us that $\underline{\Lambda}(G,\pi,\nu') < \underline{\Lambda}(G,\pi,\rho)$ for every terminal node ν' of $T(G,\pi)$. Therefore we can safely ignore $T(G,\pi,\nu)$ without miscalculating $C(G,\pi)$.

The efficiency of this technique depends mainly on two factors. On is the power of $\underline{\Lambda}$ in distinguishing between non-equivalent nodes. This, of course, can only be improved by changing Λ, which will generally involve a power/computation-time trade-off. The other factor depends on the initial labelling of G. Suppose that we wish to search the subtrees $T(G,\pi,\nu_1), \cdots, T(G,\pi,\nu_r)$, where $\nu_1, \nu_2, \cdots, \nu_r$ are the successors of ν, in the order earliest to latest. We can use the information provided by Λ by ignoring the subtree $T(G,\pi,\nu_i)$ if $\Lambda(G,\pi,\nu_i) < \Lambda(G,\pi,\nu_j)$ for some $j < i$. The number of subtrees which are thus ignored could vary from none (if the $\Lambda(G,\pi,\nu_i)$ are in non-decreasing order), to the maximum number possible (if $\Lambda(G,\pi,\nu_i) \leq \Lambda(G,\pi,\nu_1)$ for $1 \leq i \leq r$). While there is no efficient way of ensuring that the best case always occurs we can arrange for the worst case to be very unlikely. The simplest way of doing this (but not the one we will adopt) is to label G in a random fashion before commencing the generation of $T(G,\pi)$. A precise statistical analysis of how this effects the overall efficiency would be very difficult, but a rough idea can perhaps be gained from the following two theorems. We will use $E(X)$ to denote the expectation of a random variable X. The first theorem suggests that the number of ignored subtrees will not usually be much less than the maximum number possible.

2·27 **Theorem** *Let $\delta_1 < \delta_2 < \cdots < \delta_k$ be elements of a linearly ordered set Δ. Let $m_1, m_2, \cdots, m_k$ be positive integers, and put $l = m_1 + m_2 + \cdots + m_k$. Let $x_1, x_2, \cdots, x_l$ be elements of Δ, exactly m_j of which are equal to δ_j for $1 \leq j \leq k$. Now permute the x_i at random to get $x^{(1)}, x^{(2)}, \cdots, x^{(l)}$, each of the $l!$ possible permutations being equally likely. For $1 \leq i \leq l$, <u>mark</u> $x^{(i)}$ if*

$x^{(i)} \geq x^{(j)}$ *for* $j < i$, *but* $x^{(i)} \neq \delta_k$. *Let* M *be the number of marked elements. Then*

$$E(M) = \sum_{j=1}^{k-1} \frac{m_j}{1 + m_{j+1} + m_{j+2} + \cdots + m_k},$$

where the sum is taken as 0 *if* $k = 1$.

In particular, if $m_j = m$ *for* $1 \leq j \leq k$, *then*

$$E(M) = \sum_{j=1}^{k-1} \frac{m}{jm+1} \leq \log(2k).$$

□

The second theorem concerns the number of values of $\underset{\sim}{\Lambda}(G, \pi, v_i)$ for those $T(G, \pi, \nu_i)$ which are not ignored. It therefore has a bearing on the number of identity nodes which are excluded by means of Λ.

2·28 **Theorem** *Under the conditions of Theorem 2·27, let* N *be the number of different values amongst the marked elements. Then*

$$E(N) = \sum_{j=1}^{k-1} \frac{m_j}{m_j + m_{j+1} + \cdots + m_k} \leq \log l,$$

where the sum is 0 *if* $k = 1$. *In particular, if* $m_i = m$ *for* $1 \leq i \leq k$, *then* $E(N) = \sum_{j=2}^{k} j^{-1} \leq \log k$. □

An alternative to this technique for using Λ is to compute $\underset{\sim}{\Lambda}(G, \pi, \nu_i)$ for $1 \leq i \leq r$ and then only search $T(G, \pi, \nu_i)$ for those ν_i for which $\underset{\sim}{\Lambda}(G, \pi, \nu_i)$ is the largest. This is undoubtedly the best approach in many cases. However we are not adopting this method because it severely degrades the average-case behaviour. This is because the discovery of automorphisms frequently allows us to reject a subtree $T(G, \pi, \nu_i)$ without ever computing ν_i.

The theorems above relate to the effect of performing an initial random relabelling of G. The reasons we are not adopting this approach are, firstly, that this relabelling may almost double the total execution time (for a very large random graph; see Section 3·10) and, secondly, that in order to make some of the output useful (e.g. the list of

automorphisms produced) it may be necessary to translate it back to the original labelling, which is inconvenient. We will describe an alternative, but will only justify it qualitatively. A more precise analysis would be impossibly difficult to perform.

Let $\Lambda' : \underset{\sim}{G}(V) \times \underset{\sim}{\Pi}(V) \times \underset{\sim}{N}(V) \to \Delta$ be any convenient indicator function. Now devise a map $f : \Delta \to \Delta$ with the property that for pairs $x,\ y \in \Delta,\ x-y$ is very poorly correlated with $f(x)-f(y)$. (This is not meant to be a rigorous definition). For example, take $\Delta = [-1,1]$ and $f(x) = \sin(10^{10}x)$; the sign of $x-y$ is very poorly correlated with that of $f(x)-f(y)$. Now define Λ by $\Lambda(G,\pi,\nu) = f(\Lambda'(G,\pi,\nu))$. The hope is that any tendency to an unfavourable ordering of the values of $\Lambda'(G,\pi,\nu_1),\cdots,\Lambda'(G,\pi,\nu_r)$ will not occur for $\Lambda(G,\pi,\nu_1),\cdots,\Lambda(G,\pi,\nu_r)$. However, as we have stated, there is little hope of an exact statistical analysis. The best we can say is that the computational experience is favourable.

2·29 **Storage of identity nodes**

Up to this point we have been tacitly assuming that we are keeping a record of all those identity nodes so far generated, so that we can recognize later terminal nodes which are equivalent to any of them. In practice this can cause a severe storage problem, since the number of identity nodes can be very large, even if we don't count those which are eliminated by use of an indicator function. Therefore it is necessary to put a limit on the number of identity nodes (strictly, terminal nodes not known to be equivalent to an earlier node) to be stored. The optimum strategy is not clear. On the one hand, storing more identity nodes improves our chances of detecting automorphisms, which can be put to use as we have seen. On the other hand, testing two terminal nodes for equivalence is quite time consuming (especially for large graphs), and having to do a lot of these tests would have a very bad effect on the overall execution time.

The technique which we have adopted, without a great deal of theoretical justification, is to store two identity nodes at a time. The earliest terminal node ζ is always stored. The other terminal node (which may be the same as the first) is our best guess so far at the

identity node corresponding to $C(G,\pi)$. This is the node ρ referred to in Section 2·26. We also permit the algorithm to search for terminal nodes equivalent to ζ, with the aim of using the automorphisms thus discovered to shorten the total amount of work. This will sometimes degrade the performance somewhat, but on the average it works very well.

We are now able to summarize the way in which terminal nodes are processed. Suppose that we have just created a node ν, not necessarily terminal, which is not an ancestor of ζ (i.e. is later than ζ).

The node ρ and the partition θ have the same interpretation as before. Suppose that ν is the node $[\pi_1, \pi_2, \cdots, \pi_k]$ so that $|\nu| = k$. Also define $m = |\zeta|$ and $r = |\rho|$, and define variables as follows.

hh: If π_k satisfies the requirements of Lemma 2·25, then hh is the smallest value of i, $1 \leq i \leq k$, for which π_i satisfies these requirements.

Otherwise, $hh = k$.

ht: This is the smallest value of i, $1 \leq i \leq m$, for which all the terminal nodes descended from or equal to $\zeta^{(i)}$ have been shown to be equivalent.

h: The longest common ancestor of ζ and ν is $\nu^{(h)}$.

v: $\pi_{h+1} = \pi_h \perp v$

hb: The longest common ancestor of ρ and ν is $\nu^{(hb)}$.

hzb: This is the maximum value of i, $1 \leq i \leq \min\{k, r\}$, such that $\underset{\sim}{A}(G,\pi,\nu^{(i)}) = \underset{\sim}{A}(G,\pi,\rho^{(i)})$.

By *returning to* $\nu^{(i)}$ we mean backtracking in the search tree to $\nu^{(i)}$ and proceeding with the next successor of $\nu^{(i)}$ not yet generated, if any. If there are no such successors, we return to $\nu^{(i-1)}$, and so forth. *Return to* $\nu^{(0)}$ is equivalent to *stop*.

Now suppose we have just created $\nu = \nu^{(k)}$. Let $\underset{\sim}{A} = \underset{\sim}{A}(G,\pi,\nu)$.

(1) If $(k > m$ or $\underset{\sim}{A} \neq \underset{\sim}{A}(G,\pi,\zeta^{(k)})$ and $(k > r$ or $\underset{\sim}{A} < \underset{\sim}{A}(G,\pi,\rho^{(k)}))))$ go to (B).

(2) If ν is non-terminal, proceed to search $T(G,\pi,\nu)$.

(3) If $(k > m$ or $\underline{\Lambda} \neq \underline{\Lambda}(G,\pi,\zeta))$ go to (4).

If the permutation γ taking ζ onto ν is an automorphism, go to (A).

(4) If $(k > r$ or $\underline{\Lambda} < \underline{\Lambda}(G,\pi,\rho)$ or $(\underline{\Lambda} = \underline{\Lambda}(G,\pi,\rho)$ and $G(\nu) < G(\rho)))$ go to (B).

If $(\underline{\Lambda} > \underline{\Lambda}(G,\pi,\rho)$ or $(\underline{\Lambda} = \underline{\Lambda}(G,\pi,\rho)$ and $G(\nu) > G(\rho)))$ set $\rho := \nu$ then go to (B).

If $(\underline{\Lambda} = \underline{\Lambda}(G,\pi,\rho)$ and $G(\nu) = G(\rho))$ let γ be the permutation taking ρ onto ν and go to (A).

(A) {At this stage we have found an automorphism γ.}

(A1) Add $(\mathrm{fix}(\gamma), \mathrm{mcr}(\gamma))$ to Ψ (if there is room) and set $\theta := \theta \vee \theta\{\gamma\}$.

(A2) If $(v \notin \mathrm{mcr}(\theta))$ return to $\nu^{(h)}$. Otherwise, return to $\nu^{(hb)}$.

(B) {At this stage we have a terminal node ν not known to be equivalent to an earlier terminal node.}

(B1) If $(hh < k)$ add $(\mathrm{fix}(\pi_{hh}), \mathrm{mcr}(\pi_{hh}))$ to Ψ (if there is room).

(B2) Return to $\nu^{(i)}$, where $i = \min\{hh-1, \max\{ht-1, hzb\}\}$.

The only feature in the foregoing informal algorithm which we have not already justified is the use of the variable ht in Step B2. What we want to do in Step B2 is to return to the longest ancestor $\nu^{(i)}$ of ν which may conceivably have a terminal descendant which is either equivalent to ζ or improves on ρ as "the best canonical label so far". All the terminal nodes in $T(G,\pi,\nu^{(hh)})$ are known to be equivalent to ν, so we can assume that $i < hh$. Furthermore, if $i > hzb$, none of the descendants of $\nu^{(i)}$ can improve on ρ. Finally, if $i \geq ht$, and one of the descendants of $\nu^{(i)}$ was equivalent to ζ then $\nu^{(i)}$ would be equivalent to $\zeta^{(i)}$. However, all the terminal nodes descended from $\zeta^{(i)}$

are equivalent, and so all those descended from $\nu^{(i)}$ are equivalent, giving a contradiction.

2·30 We will now give a formal description of the complete algorithm.

Notes: (i) *lab* and *dig* are boolean variables. If $lab = false$, ρ is not used, and the algorithm only searches for terminal nodes equivalent to ζ. We will show in Theorem 2·33 that useful information about $\mathrm{Aut}(G)$ is still obtained. If $dig = true$, the algorithm will not use Lemma 2·25, and will be valid for digraphs and graphs with loops (for which Lemma 2·25 does not hold).

(ii) The variable ν refers everywhere to the node $[\pi_1, \pi_2, \cdots, \pi_k]$. It thus changes value if π_i $(1 \leq i \leq k)$ or k changes value.

(iii) $L \geq 1$ is an integer specifying a limit on the number of pairs $(\mathrm{fix}(x), \mathrm{mcr}(x))$ to be stored at one time. The result computed by the algorithm is independent of the choice of L, although the efficiency in general may not be.

(iv) $P \subseteq \Pi(V)$ is the set of all ordered partitions of V which satisfy the requirements of Lemma 2·25.

(v) We are assuming for convenience that $\Lambda(G, \pi, \nu)$ is real in value. If this is not the case replace "$qzb := \Lambda_k - zb_k$" by

$$qzb := \begin{cases} -1, & \text{if } \Lambda_k < zb_k \\ 0, & \text{if } \Lambda_k = zb_k \\ 1, & \text{if } \Lambda_k > zb_k \end{cases}$$

2·31 **Algorithm** *Given $G \in \mathcal{G}(V)$ and $\pi \in \Pi(V)$, find generators for $\mathrm{Aut}(G)_\pi$ and (optionally) compute $C(G, \pi)$.*

(1) $k := size := 1$

$h := hzb := index := l := 0$

$\theta :=$ discrete partition of V

$\pi_1 := \mathcal{R}(G, \pi, \pi)$

$hh := 2$

If $(\pi_1 \in P$ and $dig = false)$ $hh := 1$

If $(\pi_1$ is discrete) go to (18)

$W_1 :=$ first smallest cell of π_1

$v_1 := \min W_1$

$\Lambda_1 := e_1 := 0$

(2) $k := k + 1$

$\pi_k := \pi_{k-1} \perp v_{k-1}$

$\Lambda_k := \Lambda(G, \pi, \nu)$

If $(h = 0)$ go to (5)

If $(hzf = k - 1$ and $\Lambda_k = zf_k)$ $hzf := k$

If $(lab = false)$ go to (3)

$qzb := \Lambda_k - zb_k$

If $(hzb = k - 1$ and $qzb = 0)$ $hzb := k$

If $(qzb > 0)$ $zb_k := \Lambda_k$

(3) If $(hzb = k$ or $(lab = true$ and $qzb \geq 0))$ go to (4)

Go to (6)

(4) If $(\pi_k$ is discrete) go to (7)

$W_k :=$ first smallest cell of π_k

$v_k := \min W_k$

If $(dig = true$ or $\pi_k \notin P)$ $hh := k + 1$

$e_k := 0$

Go to (2)

(5) $zf_k := zb_k := \Lambda_k$

Go to (4)

(6) $k' := k$

$k := \min\{hh-1, \max\{ht-1, hzb\}\}$

If $(k' = hh)$ go to (13)

$l := \min\{l+1, L\}$

$\Lambda_l := \mathrm{mcr}(\pi_{hh})$

$\Phi_l := \mathrm{fix}(\pi_{hh})$

Go to (12)

(7) If $(h = 0)$ go to (18)

If $(k \neq hzf)$ go to (8)

Define $\gamma \in S_n$ by $\zeta^\gamma = \nu$

If $(G^\gamma = G)$ go to (10)

(8) If $(lab = false$ or $qzb < 0)$ go to (6)

If $(qzb > 0$ or $k < |\rho|)$ go to (9)

If $(G(\nu) > G(\rho))$ go to (9)

If $(G(\nu) < G(\rho))$ go to (6)

Define $\gamma \in S_n$ by $\nu^\gamma = \rho$

Go to (10)

(9) $\rho := \nu$

$qzb := 0$

$hb := hzb := k$

$zb_{k+1} := \infty$

Go to (6)

(10) $l := \min\{l+1, L\}$

$\Omega_l := \mathrm{mcr}(\gamma)$

$\Phi_l := \mathrm{fix}(\gamma)$

If $(\theta(\gamma) \leq \theta)$ go to (11)

$\theta := \theta \vee \theta(\gamma)$

Output γ

If $(tvc \in \mathrm{mcr}(\theta))$ go to (11)

$k := h$

Go to (13)

(11) $k := hb$

(12) If $(e_k = 1)$ $W_k := W_k \cap \Omega_l$

(13) If $(k = 0)$ *stop*

If $(k > h)$ go to (17)

If $(k = h)$ go to (14)

$h := k$

$tvc := tvh := \min W_k$

(14) If (v_k and tvh are in the same cell of θ) $index := index + 1$

$v_k := \min\{v \in W_k \mid v > v_k\}$

If $(v_k = \infty)$ go to (16)

If $(v_k \notin \mathrm{mcr}(\theta))$ go to (14)

(15) $hh := \min\{hh, k+1\}$

$hzf := \min\{hzf, k\}$

If $(lab = false$ or $hzb < k)$ go to (2)

$hzb := k$

$qzb := 0$

Go to (2)

(16) If $(|W_k| = index$ and $ht = k+1)$ $ht := k$

$size := size \times index$

←A

$index := 0$

$k := k-1$

Go to (13)

(17) If $(e_k = 0)$ set $W_k := W_k \cap \Omega_i$ for each i, $1 \leq i \leq l$, such that $\{v_1, v_2, \cdots, v_{k-1}\} \subseteq \Phi_i$

$e_k := 1$

$v_k := \min\{v \in W_k \mid v > v_k\}$

If $(v_k \neq \infty)$ go to (15)

$k := k-1$

Go to (13)

(18) $h := ht := hzf := k$

$zf_{k+1} := \infty$

$\varsigma := \nu$

$k := k-1$ ←B

If $(lab = false)$ go to (13)

$\rho := \nu$

$hzb := hb := k+1$

$zb_{k+2} := \infty$

$qzb := 0$

Go to (13)

□

2·32 Consider the stage during the execution of Algorithm 2·31 that we pass the point marked B (in Step (18)). At this instant define $K = k-1$ and $w_i = v_i$ $(1 \leq i \leq K)$.

Now let $\Gamma^{(0)} = \Gamma = \mathrm{Aut}(G)_\pi$, and define $\Gamma^{(i)} = \Gamma_{w_1, w_2, \cdots, w_i}$ (pointwise stabiliser) for $1 \leq i \leq K$. Since ς is a terminal node, the coarsest equitable partition which is finer than π and fixes $w_1, w_2, \cdots, w_K$ is discrete. Therefore $\Gamma^{(K)} = 1$.

2-33 **Theorem** *During the execution of Algorithm 2-31, each time we pass point A (in Step (16)) or point B (in Step (18)) the following are true:*

(i) $index = |\Gamma^{(k-1)}|/|\Gamma^{(k)}|$ *(point A only)*

(ii) $size = |\Gamma^{(k-1)}|$

(iii) $\theta = \theta(\Gamma^{(k-1)})$

(iv) $\Gamma^{(k-1)} = \langle Y \rangle$, *where* Y *is the set of all automorphisms output up to the present stage (in Step (10)).*

(v) $|Y| \leq n - |\theta|$

Proof: The theorem follows readily from the theory that we have already discussed, so we will only describe briefly how this needs to be assembled.

Point B is only passed once, when ζ is created, and $k = K+1$ at this stage. Point A is then passed K times, at which stages k has the values $K, K-1, \cdots, 1$ in that order.

We prove the theorem by backward induction on k. For $k = K+1$ it is obvious. Now assume it for k', for some k', $2 \leq k' \leq K+1$, and let $k = k'-1$.

Consider $\nu = [\pi_1, \pi_2, \cdots, \pi_k]$. The successors of ν, in the order earliest to latest are $\nu_1, \nu_2, \cdots, \nu_m$ where $\nu_i = \nu(w_i)$, and $W_k = \{w_1, w_2, \cdots, w_m\}$. The previous time we passed point A (or B) was when we completed our examination of the subtree $T(G, \pi, \nu_1)$. We now claim that, for $1 \leq i \leq m$, by the time we have completed examination of $T(G, \pi, \nu_i)$, w_i is in the same cell of θ as w_1 if and only if $\nu_i \sim \nu_1$.

Suppose on the contrary that there is an earliest ν_i for which our assertion is not true. If ν_i is not equivalent to ν_1 then w_i and w_1 are obviously in different cells of θ, since θ is the orbit partition of some subgroup of $\mathrm{Aut}(G)_{\pi_k}$. On the other hand, if $\nu_i \sim \nu_1$, $T(G, \pi, \nu_i)$ contains one or more terminal nodes equivalent to ζ. The nature of the algorithm is such that if one of these nodes is generated, it will be recognized as being equivalent to ζ, and if it is not generated this will only be because it has been shown to be equivalent to an earlier terminal node. Furthermore, implicit automorphisms are never used to reduce W_K, and during the examination of $T(G, \pi, \nu_i)$, if any, the only stored pairs

(Φ_j, Ω_j) which are used to reduce any W_r have $w_i \in \Phi_j$. Therefore, either w_i is already in the same cell of θ as w_1 or we are sure to discover some automorphism γ such that $\nu_i^\gamma < \nu_i$. By the induction hypothesis w_i^γ is in the same cell of θ as w_1, and so the update $\theta := \theta \vee \theta(\gamma)$ merges the cells of θ containing w_1 and w_i, contrary to hypothesis. Note also that we have just proved that $\gamma \in Y$.

We have thus concluded that the cell of θ containing w_1 is the orbit of $\Gamma^{(k-1)}$ containing w_1. Since $\theta = \theta(Y)$ by construction, and $\Gamma^{(k)} \leq \langle Y \rangle$ by the original induction hypothesis, we must have $\Gamma^{(k-1)} = \langle Y \rangle$, since $\langle Y \rangle$ contains a full set of coset-representatives for $\Gamma^{(k)}$ in $\Gamma^{(k-1)}$. This proves that $\theta = \theta(\Gamma^{(k-1)})$. The variable *index* merely counts the number of elements in the cell of θ containing w_1, so claims (i) and (ii) follow immediately.

Claim (v) follows from the simple observation that the number of cells of θ starts at n and decreases by at least one for each new element of Y. □

In closing we note a few simple properties of the set of generators of Γ found by Algorithm 2·31. These are essentially the same as those given in Theorems 36–38 in [13] and the proofs given there apply with only notational changes. Let Y be the full set of automorphisms *output* by Algorithm 2·31, and let $\Gamma = \mathrm{Aut}(G)$.

2·34 **Theorem** *(1) Y does not contain any elements of the form $\gamma\delta$, where $\gamma, \delta \in \Gamma$, $\mathrm{supp}(\gamma) \cap \mathrm{supp}(\delta) = \emptyset$ and $\gamma \neq (1) \neq \delta$.*

(2) Suppose that for some subset $Y^* \subseteq Y$, we have $\langle Y^* \rangle = \langle \Lambda^{(1)}, \Lambda^{(2)} \rangle$, where $\Lambda^{(1)}$ and $\Lambda^{(2)}$ are non-trivial subgroups of Γ with disjoint support. Then $Y^* = Y^{(1)} \cup Y^{(2)}$, where $Y^{(1)} \cap Y^{(2)} = \emptyset$, $\langle Y^{(1)} \rangle = \Lambda^{(1)}$ and $\langle Y^{(2)} \rangle = \Lambda^{(2)}$.

(3) Suppose that for some subset $W \subseteq V$ the point-wise stabiliser Γ_W has only one non-trivial orbit. Then some subset of Y generates a conjugate of Γ_W in Γ. □

IMPLEMENTATION CONSIDERATIONS

In this section we will discuss some of the problems that arise in the implementation of Algorithm 2·31 and how these have been approached. We will then examine the theoretical and empirical performance of our implementation. Finally, we will mention a few of the practical uses to which our implementation has been put.

3·1 Time versus storage

The program described in McKay [14] worked so efficiently for many classes of graphs that the practical limit on the size of graph that could be processed was set by the amount of storage available, rather than by execution time considerations. Consequently the present implementation places rather more emphasis on storage conservation, in some places to the slight detriment of time efficiency.

The variable types used by Algorithm 2·31 include graphs, sets, partitions and partition nests. We will now describe the data structures used in our implementation for each of these variable types.

3·2 Partition nests

Let $\nu = [\pi_1, \pi_2, \cdots, \pi_k] \in \mathcal{N}(V)$. Then ν can be represented by two arrays a and b of length n as follows. Define $\pi_0 = (V)$.

(i) The array a contains the elements of V in any order consistent with π_k. Precisely, if $u(a(i), \pi_k) < u(a(j), \pi_k)$ then $i < j$, for any i, $j \in V$.

(ii) Each entry of b is an integer in the interval $[0, n+1]$ chosen thus:

(a) If $u(a(i), \pi_k) = u(a(i+1), \pi_k)$, then $b(i) = n+1$ $(1 \leq i \leq n-1)$.

(b) If $u(a(i), \pi_{j-1}) = u(a(i+1), \pi_{j-1})$ but $u(a(i), \pi_j) < u(a(i+1), \pi_j)$, then $b(i) = j$ $(1 \leq j \leq k,\ 1 \leq i \leq n-1)$.

(c) $b(n) = 0$.

3·3 Unordered partitions

The only unordered partition used by Algorithm 2.31 is θ. For any $v \in V$ let θ_v denote the cell of θ containing v and let $p(v) = \min \theta_v$. Clearly θ can be uniquely represented by the array p, and most of the necessary questions about θ can be answered very quickly by reference to p. For example, if v, $w \in V$ then v and w are in the same cell of θ if and only if $p(v) = p(w)$, and $v \in \mathrm{mcr}(\theta)$ if and only if $p(v) = v$.

This representation of θ suffers from the disadvantage that updates of the form $\theta := \theta \vee \theta(\gamma)$, for $\gamma \in S_n$, are quite expensive in terms of computation time. This problem has been considerably alleviated by the use of a second array q which "chains together" the elements of each cell. More precisely, if $i \in \mathrm{mcr}(\theta)$, then $\theta_i = \{i, q(i), q(q(i)), q(q(q(i))), \cdots\}$, where the sequence terminates on the term before the first zero.

3·4 Graphs

Algorithm 3·31 requires the input graph G and, for reasonably efficient operation, requires the graph variable $G(\rho)$. From the great number of possible ways of representing these graphs in the computer, we have chosen an adjacency matrix representation because of its greater storage economy. More precisely, G is stored as a list of n bit-vectors representing $N(1, G), N(2, G), \cdots, N(n, G)$, and so requires around n^2 bits of storage. Since Algorithm 3·31 is valid also for digraphs, it is clearly not possible to reduce this storage requirement in general. However if the program was only intended to be applied on graphs with very low degree, a different sort of representation would save space, and probably time as well.

3·5 Efficiency of Algorithm 2·5

Algorithm 2·5 can easily be implemented using the data structures above. We will now consider the efficiency which can be achieved in such an implementation. The following complexity result was suggested by a related result in Gries [7]. For the necessary definitions, refer back to Section 2·9.

3·6 **Theorem** *For any $G \in \underline{G}(V)$, $\pi \in \underline{\Pi}(V)$ and distinct $v_1, v_2, \cdots, v_{m-1} \in V$, the derived partition nest $[\pi_1, \pi_2, \cdots, \pi_m]$ can be computed in $O(n^2 \log n)$ time, assuming an implementation in which $d(v, W)$ can be computed in time proportional to $|W|$, for any $v \in V$, $W \subseteq V$.*

Proof: It is obvious that the time occupied in the computation of $\pi_i \circ v_i$ for $1 \leq i \leq m-1$ and in Step (1) of Algorithm 2·5 will be $O(n^2)$. Since each execution of Step (2) of Algorithm 2·5 requires only a fixed amount of time and leads to an execution of Step (3), we are justified in restricting our attention to Step (3).

For any given W, the necessary r executions of Step (3) can be performed in $O(n|W|)$ time. Therefore the total time for the computation of $[\pi_1, \pi_2, \cdots, \pi_m]$ is $O(n^2 + n\sum|W|)$, where the sum is over all sets assigned to W during any execution of Step (2) (for any execution of Algorithm 2·5).

Let $x \in V$ and consider the real variable q_x, defined at any point of time during any execution of Algorithm 2·5 by $q_x = h_x + \log_2 l_x$. Here h_x is the number of sets containing x which have been previously assigned to W during an execution of Step (2), plus the number of sets W_j $(m \leq j \leq M)$ which contain x, plus one for the set $\{x\} = \{v_i\}$ created by the operation $\pi_i \circ v_i$, if it exists and has not already been counted. Also l_x is the current size of the cell of $\bar{\pi}$ which contains x. Note that h_x, l_x and q_x are variables which frequently change value during Algorithm 2·5.

The value of q_x clearly remains constant or decreases between different executions of Algorithm 2·5. The only other place where it can change is during Step (3), when h_x remains fixed while l_x decreases, or h_x increases by one. In the latter case l_x decreases by at least a factor of two, so that q_x does not increase. Therefore q_x is non-increasing throughout the computation, implying that its last value is bounded above by its first, which is bounded above by $2 + \log_2 n$. Therefore the final value $\bar{h}_x$ of h_x is at most $2 + \log_2 n$.

We conclude that the total time required for the computation of $[\pi_1, \pi_2, \cdots, \pi_m]$ is $O(n^2 + n\sum_{x \in V} \bar{h}_x) = O(n^2 \log n)$, as required. □

For our particular choice of data structures, and our particular implementation environment, we have found that the fastest way to compute $d(v,W)$ for $n/30 \leq |W| \leq n$ approximately is to represent W as a bit-vector and to count the number of one-bits in the bit-vector representing $N(v,G) \cap W$. Although this technique (used for $|W| > 1$) appears to reduce the total time in "the majority" of cases, it has the unfortunate side-effect of invalidating the premises of Theorem 3·6. The best replacement for the bound $O(n^2 \log n)$ which we have been able to prove is $O(n^3)$. Since the time required for the computation of $d(v,W)$ is now essentially independent of $|W|$, Step (3) of Algorithm 2·5 can be simplified by using $t = 1$. This is especially convenient if the sequence α is represented as a set of pointers to the array a (see Section 3·2).

3·7 Efficiency of Algorithm 2·31

Let $T^*(G,\pi)$ be a portion of the search tree $T(G,\pi)$ which is examined by Algorithm 2·31. Let m_1 be the number of terminal nodes of $T^*(G,\pi)$ which are equivalent to the earliest terminal node ζ (including ζ itself). Let m_2 be the number of nodes of $T^*(G,\pi)$ which are not equivalent to ζ and which do not have any descendants in $T^*(G,\pi)$. Let L be the constant defined in Section 2·30. Then the total time required by Algorithm 2·31 is $O(m_1 n^2 \log n + m_2 n^2 (L + \log n))$, under the conditions of Theorem 3·6, where m_2 may depend on L. For our implementation, this must be increased to $O(n^3(m_1 + m_2) + m_2 n^2 L)$. By Theorem 2·33, $m_1 \leq n$, but we have not found any reasonable bound on m_2. It varies in a very complicated manner with the initial labelling of the input graph and the value of L.

3·8 Other implementation details

Algorithm 2·31 has been implemented on a Cyber 170 computer, mainly in Fortran. Because of the difficulty in manipulating bit-vectors efficiently in Fortran, several small subroutines are coded in assembler language.

The indicator function Λ is evaluated by the subroutine which implements Algorithm 2·5. It is formed by taking cell sizes, relative vertex degrees and other information which is computed in the course

of Algorithm 2·5, and merging these into a single integer value in a "random" fashion (see Section 2·28).

A technique which produced considerable improvements in efficiency in some cases involves the updating of the graph $G(\rho)$ when ρ is updated. The computation of $G(\rho)$ is quite time-consuming (up to about 6 seconds for $n = 1000$), so this computation is delayed for as long as possible, in case it is not necessary.

3·9 Storage requirements

Let m be the number of machine-words required to hold a bit-vector of size n. Let K be the maximum length of a node of $T^*(G,\pi)$. Obviously $K \leq n$, but very much smaller values are normal. Define L as before. The total amount of storage required by our implementation, ignoring a minor amount independent of n, is $2mn+10n+m+(m+4)K+2mL$ words. This figure includes $2mn$ words for the storage of G and $G(\rho)$. If $lab = false$ (see Algorithm 2·31), the storage requirement is reduced by $mn+2n$ words.

3·10 Experimental performance

In figure 3·1 we give the execution time required for several families of graphs. In each description below, β gives the approximate slope of the curve in the region $50 \leq n \leq 200$. Although the results of Section 3·8 predict a value of $\beta \geq 4$, even when $m_2 = 0$, the experimental value of β is less than 3 in each of these classes.

E : empty graph on n vertices ($\beta = 2{\cdot}8$).

Q : m-dimensional cube, where $n = 2^m$ ($\beta = 2{\cdot}3$).

C : random circulant graph of degree 10 ($\beta = 2{\cdot}2$). This is defined by $V(G) = V$ and $E(G) = \{xy \mid |x-y| \in W(\mathrm{mod}\, n)\}$, where W is a random subset of $\{1, 2, \cdots, \lfloor (n-1)/2 \rfloor\}$ of size 5.

R_6 : "random" regular graph of degree 6 ($\beta = 2{\cdot}9$). There is no known practical algorithm for randomly generating regular graphs so that each graph appears with equal

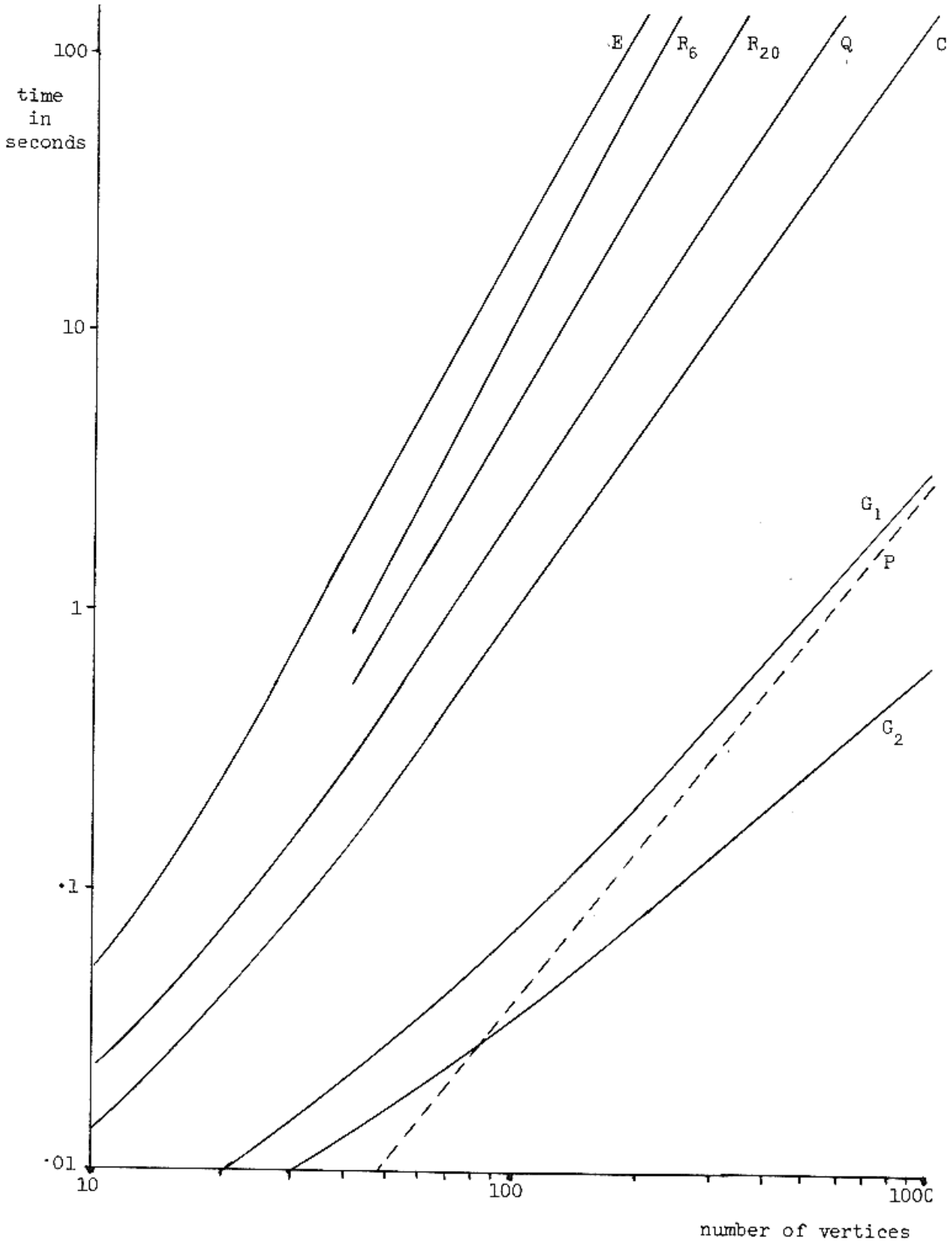
100
time
in
seconds
10
1
·1
·01
E
R6
R20
Q
C
G1
P
G2
10
100
1000
number of vertices

Figure 3·1

frequency. The graphs represented by the curve R_6 were made by randomly generating three permutations γ_1, γ_2 and $\gamma_3 \in S_n$ such that $x^\delta \neq x$ $(\delta \in \{\gamma_1^2, \gamma_2^2, \gamma_3^2\})$ and $x^{\gamma_i} \neq x^{\gamma_j}$ $(1 \leq i < j \leq 3)$ for each $x \in V$. Define G by $V(G) = V$ and $E(G) = \{xx^{\gamma_i} \mid x \in V, 1 \leq i \leq 3\}$. For $n \geq 40$ all those graphs constructed had trivial automorphism groups, and produced search trees with maximum depth 2.

R_{20} : same as R_6 but with degree 20 $(\beta = 2{\cdot}6)$.

G_1 : random graph $(\beta = 2{\cdot}0)$. Each possible edge is independently chosen or not chosen with probability $\frac{1}{2}$. The dashed line marked P in figure 5·1 gives the average time required for the computation of $G(\rho)$ for some ρ. At least one such step is essential for any program which computes $C(G, \pi)$ from G using an adjacency matrix representation. Therefore figure 3·1 suggests that the performance of our program is close to optimal for large random graphs.

G_2 : same as G_1 but with $lab = false$.

3·11 Harder examples

We have also tested our program on a number of graphs which have traditionally been regarded as difficult cases for graph isomorphism programs.

(i) The strongly regular graphs with 25 vertices required between 0·1 and 2·4 seconds, with the average time being 1·0 seconds.

(ii) A strongly regular graph G with 35 vertices can be formed from a Steiner Triple System (STS) with 15 points. The vertices of G are the blocks of the STS, and two vertices are adjacent if the corresponding blocks overlap. For the 80 graphs so formed, our program required between 0·3 and 7 seconds, with an average of 4·8 seconds. Most of these graphs have a trivial automorphism group.

(iii) Certain strongly regular graphs G with n vertices can be extended to graphs $E(G)$, having $2n+2$ vertices, which are

2-level regular (see Mathon [10]). There are good theoretical reasons [10] to expect 2-level regular graphs to be particularly difficult to process, and this is borne out by experience. The graphs A_{60} and B_{60} (60 vertices; see [10]) required 79 and 180 seconds respectively, while the graphs $A_{72} - D_{72}$ (72 vertices) required about 500 seconds each.

3·12 Design isomorphism

A *design* D (also known as a *hypergraph*) is a pair of sets (P, B), where B is a collection of subsets of P. The elements of P are called *points* and the elements of B are called *blocks*. Two designs $D_1 = (P_1, B_1)$ and $D_2 = (P_2, B_2)$ are *isomorphic* if there are bijections $f_1 : P_1 \to P_2$ and $f_2 : B_1 \to B_2$ such that $x \in X$ implies $f_1(x) \in f_2(X)$ for all $x \in P_1$ and $X \in B_1$.

Given a design $D = (P, B)$ we can construct a graph $G = G(D)$, where $V(G) = P \cup B$ and $E(G) = \{xX \mid x \in P, X \in B, x \in X\}$. It is easy to prove ([3], [17]) that two designs $D_1 = (P_1, B_1)$ and $D_2 = (P_2, B_2)$ are isomorphic if and only if there is an isomorphism $f : G(D_1) \to G(D_2)$ such that $f(P_1) = P_2$ and $f(B_1) = B_2$. Therefore Algorithm 2·31 can be used for design isomorphism.

If D is a balanced incomplete block-design (BIBD) then $G(D)$ is known to present difficulties for many graph isomorphism programs, and ours is no exception. Two 50-vertex graphs $G(D)$, named A_{50} and B_{50} in [10], required about 60 seconds each. In another experiment [18], we established the isomorphism of six BIBDs with 36 points and 36 blocks (so $n = 72$) using about 6·6 seconds of machine time each. The smallness of this figure is principally due to the reasonably rich automorphism groups of the designs.

A much more difficult problem posed by two BIBDs with 126 points and 525 blocks has been previously discussed in Stanton and McKay [17].

3·13 Hadamard equivalence

Let M_1 and M_2 be two $m \times n$ matrices with ± 1 entries. We say that M_1 and M_2 are *Hadamard equivalent* if M_2 can be obtained from

M_1 by applying an element of the group Γ generated by the following operations.

p_α : Permute the rows according to $\alpha \in S_m$.

q_β : Permute the columns according to $\beta \in S_n$.

r_i : Multiply row i by -1 $(1 \leq i \leq m)$.

c_j : Multiply column j by -1 $(1 \leq j \leq n)$.

Suppose that M is any $m \times n$ matrix with ± 1 entries. Define $G = G(M)$ to be the graph with $V(G) = \{v_i, \bar{v}_i, w_j, \bar{w}_j \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ and $E(G) = \{v_i w_j, \bar{v}_i \bar{w}_j \mid 1 \leq i \leq m, 1 \leq j \leq n, M_{ij} = 1\} \cup \{v_i \bar{w}_j, \bar{v}_i w_j \mid 1 \leq i \leq m, 1 \leq j \leq n, M_{ij} = -1\}$. We will refer to the vertices v_i and $\bar{v}_i$ as *v-type vertices*. The following theorem first appeared in McKay [16].

3·14 **Theorem** *Let $G_1 = G(M_1)$ and $G_2 = G(M_2)$. Then M_1 and M_2 are Hadamard equivalent if and only if there is an isomorphism from G_1 to G_2 which maps the v-type vertices of G_1 onto those of G_2.* □

If M is a Hadamard matrix ($m = n$ and $M^T M = nI$) then the graph $G(M)$ may prove exceedingly difficult for Algorithm 2·31. This was discovered when our implementation was applied to a collection of 126 Hadamard matrices of order 24, produced by Č. Dibley and W. D. Wallis, in an attempt to determine the equivalence classes. Several of the graphs, having large automorphism groups, were processed in about 300 seconds, but some of those with smaller automorphism groups would require more than 1800 seconds – the program was not run to completion. These graphs are all 2-level regular in the sense of Mathon [10], but are very much harder than those given in [10], even though they have larger groups. The reason for this is that the search tree $T^*(G, \pi)$ has depth 7 or 8 (compared with 4 for the graphs in [10]), although only 2 or 3 vertices generally need to be fixed in order to eliminate any non-trivial automorphisms. This means that the automorphism group is of no use for a large part of $T^*(G, \pi)$.

Other workers (see [6] for example) have found that a count of small subgraphs (e.g. cliques) can often be used to provide an initial

partitioning of the vertices of a difficult graph, which greatly speeds up a subsequent isomorphism test. Similar techniques can be used here, but they are of no use in many cases. Some of the hardest graphs amongst the 126 mentioned above have only two orbits (the v-type vertices and the others) – the initial partitioning which we were using anyway (because of Theorem 3·15). However we have devised a method based on a generalisation of the *profile* defined in [5] which can be used to refine the partitions at the immediate successors of the root node in $T^*(G,\pi)$. With this improvement, we can now process these graphs in about 20 seconds on the average.

An algorithm specifically for equivalence of Hadamard matrices has been defined by Leon [9]. The details given in [9] are insufficient to permit a direct comparison with our technique, but a cursory examination suggests that Leon's technique may be competitive with ours for this particular problem.

EXAMPLES

In this section we give two examples of the automorphism group generators produced by Algorithm 2·31. In each case we will use the notation defined in Section 2·32.

4·1 First example

In our first example G is the 5-dimensional cube defined as follows.

$$V(G) = \{(i,j,k,l,m) \mid i,j,k,l,m \in \{0,1\}\}$$
$$E(G) = \{(i_1,j_1,k_1,l_1,m_1)(i_2,j_2,k_2,l_2,m_2) \mid (i_1-i_2)^2+(j_1-j_2)^2 + (k_1-k_2)^2+(l_1-l_2)^2+(m_1-m_2)^2 = 1\}$$

The elements of $V(G)$ are numbered $1,2,\cdots,32$ in lexicographic order.

For this graph we find $K = 5$, $w_1 = 1$, $w_2 = 16$, $w_3 = 24$, $w_4 = 28$ and $w_5 = 30$. The output produced is as below. The execution time was 0·18 seconds.

(3 5) (4 6) (11 13) (12 14) (19 21) (20 22) (27 29) (28 30)
$|\Gamma^{(4)}| =$ 2 $|\theta(\Gamma^{(4)})| =$ 24

(2 3) (6 7) (10 11) (14 15) (18 19) (22 23) (26 27) (30 31)

$|\Gamma^{(3)}| =$ 6 $|\theta(\Gamma^{(3)})| =$ 16

(5 9) (6 10) (7 11) (8 12) (21 25) (22 26) (23 27) (24 28)

$|\Gamma^{(2)}| =$ 24 $|\theta(\Gamma^{(2)})| =$ 10

(9 17) (10 18) (11 19) (12 20) (13 21) (14 22) (15 23) (16 24)

$|\Gamma^{(1)}| =$ 120 $|\theta(\Gamma^{(1)})| =$ 6

(1 2) (3 4) (5 6) (7 8) (9 10) (11 12) (13 14) (15 16) (17 18) (19 20)
(21 22) (23 24) (25 26) (27 28) (29 30) (31 32)

$|\Gamma| =$ 3840 $|\theta(\Gamma)| =$ 1

4·2 Second example

In our second example G is the lexicographic product $C_5[C_5]$ defined as follows.

$$V(G) = \{(i, j) \mid 1 \leq i, j \leq 5\}$$
$$E(G) = \{(i_1, j_1)(i_2, j_2) \mid |i_1 - i_2| \equiv 1 (\mathrm{mod}\, 5) \text{ or } i_1 = i_2 \text{ and } |j_1 - j_2| \equiv 1 (\mathrm{mod}\, 5)\}$$

The vertices are labelled $1, 2, \cdots, 25$ in lexicographic order.

For this graph we find $K = 10$, $w_1 = 1$, $w_2 = 3$, $w_3 = 11$, $w_4 = 13$, $w_5 = 16$, $w_6 = 18$, $w_7 = 21$, $w_8 = 23$, $w_9 = 6$ and $w_{10} = 8$. The output below was generated in 0.23 seconds.

(7 10) (8 9)

$|\Gamma^{(9)}| =$ 2 $|\theta(\Gamma^{(9)})| =$ 13

(6 7 8 9 10)

$|\Gamma^{(8)}| =$ 10 $|\theta(\Gamma^{(8)})| =$ 21

(22 25) (23 24)

$|\Gamma^{(7)}| =$ 20 $|\theta(\Gamma^{(7)})| =$ 19

(21 22 23 24 25)

$|\Gamma^{(6)}| =$ 100 $|\theta(\Gamma^{(6)})| =$ 17

(17 20) (18 19)

$|\Gamma^{(5)}| =$ 200 $|\theta(\Gamma^{(5)})| =$ 15

(16 17 18 19 20)

$|\Gamma^{(4)}| =$ 1000 $|\theta(\Gamma^{(4)})| =$ 13

(12 15) (13 14)

$|\Gamma^{(3)}| =$ 2000 $|\theta(\Gamma^{(3)})| =$ 11

(11 12 13 14 15)

(6 21) (7 22) (8 23) (9 24) (10 25) (11 16) (12 17) (13 18) (14 19) (15 20)

$|\Gamma^{(2)}| =$ 20000 $\quad |\theta(\Gamma^{(2)})| =$ 7

(2 5) (3 4)

$|\Gamma^{(1)}| =$ 40000 $\quad |\theta(\Gamma^{(1)})| =$ 5

(1 2 3 4 5)

(1 6 11 16 21) (2 7 12 17 22) (3 8 13 18 23) (4 9 14 19 24) (5 10 15 20 25)

$|\Gamma| =$ 1000000 $\quad |\theta(\Gamma)| =$ 1

REFERENCES

[1] V.L. Arlazarov, I.I. Zuev, A.V. Uskov and I.A. Faradzev: An algorithm for the reduction of finite non-oriented graphs to canonical form. *Zh. vȳchisl. Mat. mat. Fiz.* 14, 3 (1974) 737–743.

[2] T. Beyer and A. Proskurowski: Symmetries in the graph coding problem. *Proc. NW76 ACM/CIPC Pac. Symp.* (1976) 198–203.

[3] C. Bohm and A. Santolini: A quasi-decision algorithm for the p-equivalence of two matrices. *ICC BULLETIN* 3, 1 (1964) 57–69.

[4] J.A. Bondy and U.S.R. Murty: *Graph Theory with Applications*, Macmillan (1976).

[5] J. Cooper, J. Milas and W.D. Wallis: Hadamard equivalence. International Conference on Combinatorial Mathematics, Canberra (1977), Lecture Notes in Mathematics 686, Springer-Verlag 126–135.

[6] D.G. Corneil and R.A. Mathon: Algorithmic techniques for the generation and analysis of strongly regular graphs and other combinatorial configurations. *Annals of Discrete Math.* 2 (1978) 1–32.

[7] D. Gries: Describing an algorithm by Hopcroft. *Acta Informatica* 2 (1973) 97–109.

[8] J. Hopcroft: An $n \log n$ algorithm for minimizing states in a finite automaton. *Theory of Machines and Computations.* Academic Press (1971) 189–196.

[9] J.S. Leon: An algorithm for computing the automorphism group of a Hadamard matrix. *J. Combinatorial Theory* (A) 27 (1979) 289–306.

[10] R. Mathon: Sample graphs for isomorphism testing. *Proc. 9th. Southeastern Conf. on Comb., Graph Theory and Computing* (1978), to appear.

[11] R. Mathon: Personal communication.

[12] B.D. McKay: *Backtrack programming and the graph isomorphism problem.* M. Sc. Thesis, University of Melbourne (1976).

[13] B.D. McKay: Backtrack programming and isomorph rejection on ordered subsets. *Ars Combinatoria* 5 (1978) 65–99.

[14] B.D. McKay: Computing automorphisms and canonical labellings of graphs. International Conference on Combinatorial Mathematics, Canberra (1977), Lecture Notes in Mathematics 686, Springer-Verlag, 223–232.

[15] B.D. McKay: *Topics in Computational Graph Theory.* Ph. D. Thesis, University of Melbourne (1980).

[16] B.D. McKay: Hadamard equivalence via graph isomorphism. *Discrete Math.* 27 (1979) 213–214.

[17] B.D. McKay and R.G. Stanton: Isomorphism of two large designs. *Ars Combinatoria* 6 (1978) 87–90.

[18] B.D. McKay and R.G. Stanton: Some graph isomorphism computations. *Ars Combinatoria,* 9 (1980) 307–313.

[19] H. Wielandt: *Finite Permutation Groups.* Academic Press (1964).