COMPUTING AUTOMORPHISMS AND CANONICAL LABELLINGS OF GRAPHS

Brendan D. McKay
Department of Mathematics,
University of Melbourne,
Parkville, Victoria, 3052
Australia

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 $\underset{\sim}{G}(V)$ to be the set of all (labelled) graphs with vertex set V. Let S_n be the symmetric group acting on V. For $G \in \underset{\sim}{G}(V)$ and $g \in S_n$, define $G^g \in \underset{\sim}{G}(V)$ 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, Aut(G), is the group $\{g \in S_n \mid G^g = G\}$.

The *canonical label problem* is to find a map *canon*: $\underset{\sim}{G}(V) \to \underset{\sim}{G}(V)$ such that for $G \in \underset{\sim}{G}(V)$ and $g \in S_n$,

(1) *canon* (G) is isomorphic to G, and (2) *canon* (G^g) = *canon* (G).

Note that there may be many functions *canon* satisfying (1) and(2).

If $G,H \in \underset{\sim}{G}(V)$, we see that G and H are isomorphic if and only if *canon* (G) = *canon* (H).

In this paper we present a new algorithm for computing *canon* (G) which will also find a set of fewer than n automorphisms which generate 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 π of disjoint non-empty subsets of V whose union is V. The elements of π 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 $\underset{\sim}{\Pi}(V)$ respectively.

Define $\Pi^*(V) = \Pi(V) \cup \underset{\sim}{\Pi}(V)$. Let $\pi_1, \pi_2 \in \Pi^*(V)$. We write $\pi_1 \leq \pi_2$ (π_1 is *finer* than π_2, π_2 is *coarser* than π_1) if every cell of π_1 is contained in some cell of π_2. If both $\pi_1 \leq \pi_2$ and $\pi_2 \leq \pi_1$, we write $\pi_1 \cong \pi_2$. If $\pi \in \Pi^*(V)$, the number of cells of π is denoted by $|\pi|$. π 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* π. Denote by $\pi \vee g$ the finest partition of V which is coarser than π 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 $G \in \underset{\sim}{G}(V)$. 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 $\pi \in \Pi^*(V)$. π 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 π, the coarsest equitable partition which is finer than π will be denoted by $\xi(\pi)$. Similarly, $\theta(\pi)$ denotes the partition whose cells are the orbits of the subgroup of Aut(G) which fixes π. 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 π is equitable, and $n - |\pi| \leq 5$, the smallest cells of π of size ≥ 2 are cells of $\theta(\pi)$. (Not true if G is a digraph.) □

Corneil proved in [2] that for any π, $\theta(\pi) = \xi(\pi)$ 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 α be a subset of π.

ALGORITHM 1: Compute $\tilde{\pi} = \mathcal{R}(G, \pi, \alpha)$

(1) $\tilde{\pi} \leftarrow \pi$

(2) If $\alpha = \emptyset$ or $\tilde{\pi}$ is discrete, *stop*.
Choose any non-null subset β of α.
$\alpha \leftarrow \alpha \backslash \beta$, $i \leftarrow 1$
(Suppose $\tilde{\pi} = \{C_1, C_2, \ldots, C_k\}$ 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 $v \in C_i$.
$\tilde{\pi} \leftarrow \tilde{\pi} \cup \{D_1, D_2, \ldots, D_s\} \backslash \{C_i\}$

$\alpha \leftarrow \alpha \cup \{D_2, \ldots, D_s\}$ (if $s \geq 2$)

$i \leftarrow i + 1$

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

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

Proof: (a) In step (3) of the algorithm, $|\alpha|$ is not increased unless $\tilde{\pi}$ is made finer. Since $|\alpha|$ 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 $\xi(\pi) \leq \tilde{\pi}$. Since $\beta \subseteq \tilde{\pi}$, each element of β is a union of cells of $\xi(\pi)$. Hence we will also have $\xi(\pi) \leq \tilde{\pi}$ 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 $\tilde{\pi}$ is made successively finer by the algorithm, v and w must always be in the same cell of $\tilde{\pi}$.

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

(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 α.

(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 $\mathcal{R}(G, \pi, \pi) \simeq \xi(\pi)$. □

A considerable advantage which Algorithm 1 has over previous algorithms is that in some important situations, α can be a proper subset of π. Suppose π_1 is an equitable partition of V coarser than π. Let $\alpha \subseteq \pi$ be such that for any $D \in \pi_1$, $C \subseteq D$ for at most one $C \in \pi \backslash \alpha$. Then it can be proved (see [3]) that $\mathcal{R}(G, \pi, \alpha) \simeq \xi(\pi)$.

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 *canon* (G) and generators for 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 π is discrete, it is an end-node of the tree and we will call it a *terminal partition*. If π is not discrete, let C_i be the first cell of π with the smallest size ≥ 2. For $v \in C_i$ define $\pi \circ v = (C_1, \ldots, C_i \setminus \{v\}, \ldots, C_r, \{v\})$. The successors of π in the search tree are the partitions $\mathcal{R}(G, \pi \circ v, \{v\})$, for $v \in C_i$, 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^τ by labelling the vertices of G in the order they appear in τ. 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 $\varepsilon \in X$, $\mathrm{Aut}(G) = \{g \mid \tau = \varepsilon^g, \tau \in X, \tau \sim \varepsilon\}$. Furthermore, by comparing the adjacency matrices lexicographically we can define an order on $\underset{\sim}{G}(V)$ and define *canon* $(G) = \max\{G^\tau \mid \tau \in X\}$.

Since $|X|$ is a multiple of $|\mathrm{Aut}(G)|$ 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 $G \in \underset{\sim}{G}(V)$ and $\pi \in \underset{\sim}{\Pi}(V)$ such that $\Lambda(G^g, \pi^g) = \Lambda(G, \pi)$ for any $g \in S_n$. An example of such a function would be the number of edges of G whose end-points are both in the same cell of π. 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 τ. Then τ can be associated with the sequence $\underset{\sim}{\Lambda}(\tau) = (\Lambda(G, \pi_1), \Lambda(G, \pi_2), \ldots, \Lambda(G, \pi_k))$. Clearly τ_1 and τ_2 cannot be equivalent unless $\underset{\sim}{\Lambda}(\tau_1) = \underset{\sim}{\Lambda}(\tau_2)$. Further efficiency can be achieved by ordering the vectors $\underset{\sim}{\Lambda}(\tau)$ lexicographically and redefining *canon* (G) to be $\max\{G^\tau \mid \tau \in X, \underset{\sim}{\Lambda}(\tau) = \Lambda^*\}$, where $\Lambda^* = \max\{\underset{\sim}{\Lambda}(\tau) \mid \tau \in X\}$. By this means we can eliminate sections of the search tree which cannot contain either new automorphisms or *canon* (G).

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

4. THE ALGORITHM

ALGORITHM 2: Compute generators for Aut(G), where G is a graph or digraph, and optionally compute *canon* $(G) = G^\beta$.

Notes: (i) Lower case Greek letters represent partitions. The variables C_1, C_2, ... represent sets.

(ii) The minimum of an empty set is defined to be ∞, where ∞ is a number larger than any number it is compared to.

(iii) If x is a permutation (or partition), $\Omega(x)$ 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) $k \leftarrow 1$, size $\leftarrow 1$, $h \leftarrow 0$, index $\leftarrow 0$
$\theta \leftarrow$ discrete partition of V
$\pi_1 \leftarrow \mathcal{R}(G, (V), (V))$
If *dig* or $n - |\pi_1| \geq 6$, $q \leftarrow 2$, else $q \leftarrow 1$.
If π_1 is discrete go to (20).
$C_1 \leftarrow$ first cell of π_1 with smallest size ≥ 2
$e_1 \leftarrow 0$, $v_1 \leftarrow \min C_1$, $w_1 \leftarrow v_1$

(2) $k \leftarrow k + 1$
$\pi_k \leftarrow \mathcal{R}(G, \pi_{k-1} \circ v_{k-1}, \{v_{k-1}\})$
$z \leftarrow \Lambda(G, \pi_k)$
If π_k is not discrete, $e_k \leftarrow 0$ and $C_k \leftarrow$ first cell of π_k with smallest size ≥ 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 π_k is discrete, go to (7).
$v_k = \min C_k$
If $h = 0$, $w_k \leftarrow v_k$.
If *dig* 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 $\varepsilon^g = \pi_k$.

(8) ($g \in \mathrm{Aut}(G)$: Write g if desired, and store $(\Phi(g), \Omega(g))$ if room is available.)
$\theta \leftarrow \theta \vee g$, $k \leftarrow h$

(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 θ, index $\leftarrow$ index + 1.
$v_k \leftarrow \min \{v \in C_k \mid v > v_k\}$
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).
$hy \leftarrow k$, $qy \leftarrow 0$
Go to (2).

(12) size $\leftarrow$ size $\times$ index
index $\leftarrow 0$, $k \leftarrow k-1$
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) $v_k \leftarrow \min\{v \in C_k \mid v > v_k\}$
If $v_k = \infty$, set $k \leftarrow k-1$ and go to (9).
Go to (11).

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

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

(17) $\beta \leftarrow \pi_k$, $hy \leftarrow k$, $hb \leftarrow k$, $y_{k+1} \leftarrow \infty$, $qy \leftarrow 0$

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

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

(20) $h \leftarrow k$, $hx \leftarrow k$, $x_{k+1} \leftarrow \infty$, $\varepsilon \leftarrow \pi_k$
$k \leftarrow k+1$
If not *lab*, go to (9).
$\beta \leftarrow \pi_{k-1}$, $hy \leftarrow k+1$, $hb \leftarrow k+1$, $y_{k+2} \leftarrow \infty$, $qy \leftarrow 0$
Go to (9).

Let G be a graph or a digraph and let $A = \mathrm{Aut}(G)$. If $W \subseteq V$, A_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 $\theta^{(k-1)}$, $\mathrm{index}^{(k-1)}$, $\mathrm{size}^{(k-1)}$ to be the current values of θ, index and size, and let $\Sigma^{(k-1)}$ be the set of all elements of $\mathrm{Aut}(G)$ 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) $\mathrm{size}^{(k)} = |A^{(k)}|$
(ii) $\mathrm{index}^{(k)} = |A^{(k)}|/|A^{(k+1)}|$ $\quad (k < K)$
(iii) the cells of $\theta^{(k)}$ are the orbits of $A^{(k)}$
(iv) $\Sigma^{(k)}$ generates $A^{(k)}$
(v) $|\Sigma^{(k)}| \leq n - \ell_k$, where $A^{(k)}$ has ℓ_k orbits.

(2) If *lab* is true, G^{β} 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 Aut(G) from $\Sigma^{(0)}$ is given in [4], as are a few other facts about $\Sigma^{(0)}$, for example the following lemma.

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. 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 π_i 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 ℓ is the maximum value of k for which π_k is ever computed (obviously, $\ell \leq n$). Then at most $n(2m + 8) + 2\ell m$ words of storage are required by the program, plus an extra $n(m + 2)$ words if *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 π and from various items remaining from the computation of π 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 *canon* (G) as well as 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.

$\overline{K}_n$: empty graph.
RD : randomly selected digraph with constant out-degree = $\frac{1}{2}n$.
RC : randomly selected circulant graph with degree = $\frac{1}{2}n$.
Q_m : m-dimensional cube; $n = 2^m$.
RG : randomly selected graph with edge-density = $\frac{1}{2}$
(1): *canon* (G) found, (2): *canon* (G) not found.
SR25 : strongly regular graphs on 25 vertices (average time).
SR35 : strongly regular block intersection graphs of Steiner triple systems with 15 points and 35 blocks (average time).

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

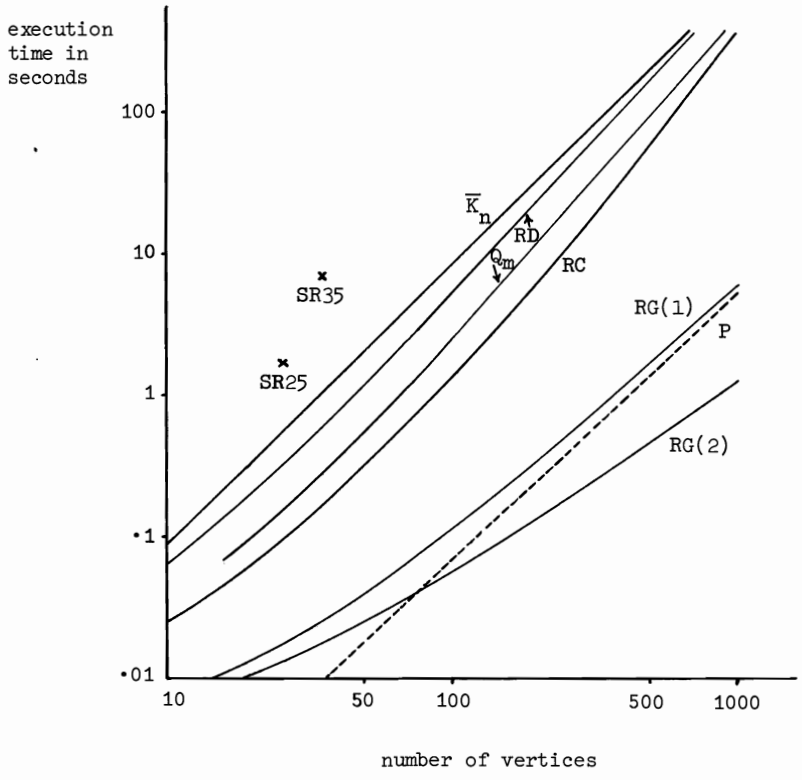

Figure 1

step in any program which computes *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 π. 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 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) 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. Aut(G) is transitive and has order 200. The time taken was 0·16 seconds.

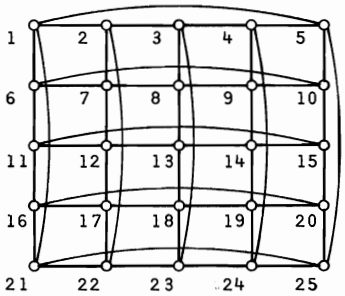

Figure 2

REFERENCES

[1] M. Behzad and G. Chartrand, *Introduction to the theory of graphs*, Allyn and Bacon, Boston (1971).

[2] D.G. Corneil, *Graph Isomorphism*, Ph.D. Thesis, Univ. of Toronto (1968).

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

[4] B.D. McKay, "Backtrack programming and isomorph rejection on ordered subsets", to appear in *Proc. 5th Australian Conf. on Combin. Math.* (1976).

[5] R. Parris, *The coding problem for graphs*, M.Sc. Thesis, Univ. of West Indies (1968).

[6] J.P. Steen, "Principle d'un algorithme de recherche d'un isomorphisme entre deux graphes", *RIRO*, R-3, 3 (1969), 51-69.

[7] H. Wielandt, *Finite permutation groups*, Academic Press, New York and London (1964).