BACKTRACK PROGRAMMING

AND THE

GRAPH ISOMORPHISM PROBLEM

BY

BRENDAN DAMIEN McKAY

A THESIS SUBMITTED FOR THE DEGREE OF

MASTER OF SCIENCE

AT THE

UNIVERSITY OF MELBOURNE

JULY 1976

NOTES 7/7/77

(a) Fig. 5.2 may be incomplete.

(b) p.143 : The Hoffman-Singleton graph on 50 points is an example of a transitive graph with $M_2 \neq 0$.

(c) Fig. 9.4 : Execution times are now considerably better for edge-sparse graphs. For random graphs, new time $\doteq$ old time $\times \sigma \times 1.05$.

(d) 9.53 : The assertions in this section are not in general true.

Suppose $\zeta = [C_1|C_2|\cdots|C_k]$ and define $\Psi = \{\gamma \in \Gamma(G) | C_i^\gamma \cap C_i \neq \emptyset, \ 1 \leq i \leq k\}$.

It is easy to show that the set Y of all elements of Γ found by 9.21 or 9.24 lies in Ψ, but in general it may not be in Γ_ζ. However, if $Y \leq \Gamma_\zeta$, then $\langle Y \rangle = \Gamma_\zeta$.

Γ_ζ will be found correctly if $\Psi = \Gamma_\zeta$ (example : all but one of the cells of ζ are trivial), if $M_2 = 0$ (for 9.21, not 9.24) or under various other conditions.

In practice, Γ_ζ and $f(G, \zeta)$ can be determined by extending G with a few extra vertices in the right way.

PREFACE

This thesis originally arose from the need for an algorithm suitable for canonically labelling a graph with a large automorphism group $\Gamma(G)$. Since all the existing algorithms that we knew of had execution times highly dependent on $|\Gamma(G)|$, an effort was made to devise a program which did not suffer from this deficiency. Eventually, a system was devised by which elements of $\Gamma(G)$ could be found during the labelling process and used to reduce the amount of work. It soon became evident that our algorithm was ideal for the study of $\Gamma(G)$, since it appeared to find only a small set (less than $|V(G)|$) of generators for $\Gamma(G)$.

When it came to constructing rigorous proofs for the correctness of our algorithm, it became immediately obvious that a more general setting was possible. Very soon a theory had emerged of backtrack programming of a certain type and of the *invariance group* of such a program. This theory is presented in Chapters Six and Seven. Except as stated there, it is all original.

In earlier chapters we develop the necessary groundwork. Chapters One to Three are devoted to the elementary concepts of permutation groups, graphs, lattices, partitions and various other objects. In Chapter Four we introduce the lattice $\Theta(\Psi)$ of partitions defined by the orbits of subgroups of Ψ. In Chapter Five we treat a related lattice $\Xi(G)$ of equitable partitions of the point-set of a graph G. The relationship between $\Xi(G)$ and $\Theta(\Gamma(G))$ is considered, and a new algorithm for finding the coarsest equitable partition finer than a given partition is presented.

In Chapter Eight we give a reasonably general treatment of existing solutions to various "graph isomorphism problems". This

treatment is probably new. We then concentrate on the problem of canonically labelling a graph, and devise a general method of solution which appears to include most existing algorithms.

In Chapter Nine, we present several versions of our own algorithm for canonically labelling a graph. We show that it falls into the general class described in Chapter Eight but differs in that the methods developed in Chapter Seven have been applied. We demonstrate that the algorithm finds a set of no more than $|V(G)| - p$ generators for $\Gamma(G)$, where $\Gamma(G)$ has p orbits. The efficiency of the algorithm is then examined. For large random graphs we claim that it is impossible to devise an algorithm which is very much faster.

There are many people without whose help this thesis might have been considerably more difficult to complete. Special thanks are due to Dr. B.D. Craven for his many efforts, and to Dr. D.A. Holton for his detailed criticisms of the manuscript. I would also like to acknowledge Mr. Chris Godsil for helping in the practical evaluation of the program and for the many spirited discussions which kept my enthusiasm alive. Finally, thanks are due to Miss Joan Beverley for helping to read the proofs and to Mrs. Ann Windsor for her excellent typing.

CONTENTS

CHAPTER ONE

INTRODUCTION

1.1 In this chapter we introduce some of the basic concepts from the theories of matrices, permutation groups and graphs.

1.2 If Δ is any set, $|\Delta|$ is the cardinality of Δ and 2^{Δ} is its power set. The null set will be denoted ϕ. If Δ_1 and Δ_2 are sets, the set difference of Δ_1 and Δ_2 is denoted $\Delta_1 \backslash \Delta_2 = \{x | x \in \Delta_1, \text{ but } x \notin \Delta_2\}$. The cartesian product of Δ_1 and Δ_2 is denoted $\Delta_1 \times \Delta_2$. The symbol *iff* is an abbreviation for "if and only if".

To avoid confusion with our notation for permutations, a sequence (or vector) of elements of Δ will be denoted $[x_1, x_2, \cdots, x_r]$. The sequence with no elements is the *null sequence*, and denoted $[]$.

Let $f(n)$ and $g(n)$ be real-valued functions defined for positive integers n. If there is a constant M so that $|f(n)| \leq M|g(n)|$ for $n > 0$, we write $f(n) = O(g(n))$.

1.3 Let A and B be matrices. The entry in the i-th row and j-th column of A is denoted A_{ij}. The transpose and inverse (if it exists) of A are respectively denoted A' and A^{-1}. The *tensor product* $A \otimes B$ of A and B is defined as follows. Suppose A is $n \times m$. Then $A \otimes B$ consists of n rows of m blocks, the j-th block in the i-th row being the matrix $A_{ij}B$. The basic properties of the tensor product can be found in Lancaster [38], but we will only have need for the definition.

1.4 Let V be a finite set. A *permutation* of V is a bijection from V onto itself. The set of all permutations of V is denoted S(V),

or S_n if $V = \{1, 2, \cdots, n\}$. If $\gamma \in S(V)$ and $v \in V$, the image of v under γ is denoted v^γ. Similarly, if $\Omega \subseteq S(V)$, v^Ω is the set $\{v^\gamma | \gamma \in \Omega\}$. More generally, if u^γ is defined for $u \in U$ and $\gamma \in \Omega \subseteq S(V)$, we define $U^\Omega = \{u^\gamma | u \in U, \gamma \in \Omega\}$.

Permutations will be written using the familiar cyclic notation. Thus if $V = \{1, 2, 3, 4, 5, 6\}$ and $[1^\gamma, 2^\gamma, 3^\gamma, 4^\gamma, 5^\gamma, 6^\gamma] = [2, 1, 4, 5, 3, 6]$, γ can be written as $(1\ 2)(3\ 4\ 5)$. In this case $(1\ 2)$, $(3\ 4\ 5)$ and (6) are called the *cycles* of γ. *Trivial* (unit-length) cycles, like (6) are commonly omitted from the notation. The points (elements) of V they contain are said to be *fixed* by the permutation. The identity map on V, which fixes every point of V, is called the *identity* or *trivial* permutation, and denoted (1). A permutation of the form $(v_1\ v_2)$, where $v_1, v_2 \in V$ is called a *transposition*.

1.5 Two permutations can be *multiplied* in the manner usual for map composition. Thus if $v \in V$ and $\gamma, \delta \in S(V)$ we have $\gamma\delta \in S(V)$ where $v^{\gamma\delta} = (v^\gamma)^\delta$. Under this operation $S(V)$ forms a group, called the *symmetric group on* V. We can now define a *permutation group on* V as a subgroup of $S(V)$. The smallest such group is the *trivial group* $\{(1)\}$. The theory of permutation groups additional to what we give here can be found in Wielandt [79] or Scott [64].

If $\Psi \leq S(V)$ and $v \in V$, then v^Ψ is called an *orbit* of Ψ. It is easy to see that every point of V is in some orbit (since $(1) \in \Psi$) and that no two orbits overlap. If Ψ has just one orbit it is called *transitive*.

1.6 If $\Psi \leq S(V)$, $U \subseteq V$ and $U^\Psi = U$, then Ψ induces a group of

permutations on U, which we denote $\Psi|_U$. Again, if $U \subseteq V$ we can define the *(point-wise) stabiliser of* U *in* Ψ to be the group $\Psi_U = \{\gamma \in \Psi | u^\gamma = u$ for all $u \in U\}$. We will find it convenient to write Ψ_v instead of $\Psi_{\{v\}}$, $\Psi_{v,w}$ instead of $\Psi_{\{v,w\}}$ and so on.

If $\Omega \subseteq S(V)$, the group *generated by* Ω is defined to be the smallest subgroup $\langle \Omega \rangle$ of S(V) which contains Ω. In particular, $\langle \phi \rangle = \{(1)\}$.

If $\Omega \subseteq S(V)$ and $\gamma \in S(V)$ we define $\gamma\Omega = \{\gamma\delta | \delta \in \Omega\}$ and similarly $\Omega\gamma = \{\delta\gamma | \delta \in \Omega\}$. If $\Psi \leq S(V)$ and $\Omega_1, \Omega_2 \subseteq \Psi$ we say that Ω_1 and Ω_2 are *conjugate* in Ψ if $\Omega_2 = \gamma^{-1}\Omega_1\gamma$ for some $\gamma \in \Psi$. Conjugacy forms an equivalence relation on the power set 2^Ψ and partitions 2^Ψ into *conjugacy classes*.

1.7 Suppose $\Psi, \Lambda \leq S(V)$ and any point of V not fixed by Ψ is fixed by Λ. Then the permutation group $\langle \Psi \cup \Lambda \rangle$ will be called the *direct sum of* Ψ *and* Λ and denoted $\Psi \oplus \Lambda$.

Suppose that $V = X \times Y$, where $X = \{x_1, \cdots, x_m\}$ and $Y = \{y_1, \cdots, y_m\}$. Let $\Psi \leq S(X)$ and $\Lambda \leq S(Y)$. The *wreath product* $\Psi[\Lambda]$ is a permutation group on V defined as follows. Each element γ of $\Psi[\Lambda]$ corresponds to a sequence $[\alpha, \beta_1, \cdots, \beta_m]$ where $\alpha \in \Psi$ and $\beta_i \in \Lambda$ $(1 \leq i \leq m)$. The action of γ on V is defined by $(x_i, y_j)^\gamma = (x_i^\alpha, y_j^{\beta_i})$ for $(x_i, y_j) \in V$. If we set $\alpha = (1)$ and $\beta_i = (1)$ for $i \neq k$, for fixed k, we find a subgroup of $\Psi[\Lambda]$ isomorphic to Λ, which we will call a *copy of* Λ *in* $\Psi[\Lambda]$.

1.8 In general, our graph-theoretic notation will follow that of Behzad and Chartrand [4], and any definitions we have inadvertantly

omitted can be found in that book. A *directed graph (digraph)* G consists of a finite non-empty set $V = V(G)$ and a set $E(G)$ of ordered pairs of distinct elements of V. Elements of $V(G)$ and $E(G)$ are respectively called the *points* and *directed edges* of G. A *graph* G consists of a finite non-empty set $V = V(G)$ and a set $E(G)$ of unordered pairs of distinct elements of V. Elements of $V(G)$ and $E(G)$ are respectively called the *points* and *edges* of G. If $\{v_1, v_2\} \in E(G)$ we can say that the point v_1 is *connected* to the point v_2, or alternatively that v_1 and v_2 are *adjacent*. We can also say that the point v_1 and the edge $\{v_1, v_2\}$ are *incident*.

1.9 Two graphs are *isomorphic* if there is a bijection $\psi : V(G_1) \to V(G_2)$ which preserves adjacency. A *labelled graph* is a graph whose points are associated in a 1-1 fashion with a set of distinct labels. We will not always maintain a concise distinction between graphs and labelled graphs in this thesis for the reasons which follow. Almost invariably, we have used the set $V = \{1, 2, \cdots, n\}$ both for the point-set of a graph and for the labels of a labelled graph. A graph with V as its point-set can be considered labelled if we think of the point v being labelled with the number v. Similarly, a labelled graph whose points have been labelled with the numbers $\{1, 2, \cdots, n\}$ can be thought of as corresponding to a graph whose points are the labels of the labelled graph. In general, we will use the adjective "labelled" when we wish to emphasize that the properties we are considering may not be preserved under a re-labelling, or that we are taking a *particular* graph G with points $\{1, 2, \cdots, n\}$ rather than any graph isomorphic to G. Thus when we define $\underset{\sim}{G}(V)$ to be the set of all labelled graphs with point-set V we mean that isomorphic but non-identical graphs are to be considered distinct elements of $\underset{\sim}{G}(V)$.

1.10 Let $G \in \underline{G}(V)$ and $\gamma \in S(V)$. We define G^γ to be the graph with point-set V such that $\{v_1^\gamma, v_2^\gamma\} \in E(G^\gamma)$ iff $\{v_1, v_2\} \in E(G)$. Obviously G and G^γ are isomorphic. If they are actually identical we say that γ is an *automorphism* of G. The set of all automorphisms of G form a group called the *automorphism group* of G and denoted $\Gamma(G)$. G is said to be *transitive* if $\Gamma(G)$ is.

1.11 A graph H is called a *subgraph* of the graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $V(H) = V(G)$, H is called a *spanning subgraph* of G. If $U \subseteq V(G)$ and $U \neq \phi$ the subgraph $\langle U \rangle$ of G *induced* by U has point-set U and all edges of G incident with two elements of U.

1.12 Several important families of graphs are given special names. The *complete graph* on n points, K_n, has every pair of points adjacent. K_3 is also called a *triangle*. The *cycle* on n points, Z_n, has $V(Z_n) = \{v_1, \cdots, v_n\}$ and $E(Z_n) = \{\{v_i, v_j\} | i - j \equiv 1 \pmod n\}$. We are avoiding the more common notation C_n since this will be used for the cells of a partition. Finally, the path on n points, P_n, has $V(P_n) = \{v_1, \cdots, v_n\}$ and $E(P_n) = \{\{v_i, v_{i+1}\} | 1 \leq i < n\}$. The points v_1 and v_n and the *endpoints* of P_n and the *length* of P_n is n - 1.

A subgraph of G isomorphic to P_n for some $n \geq 1$ is called a *path* in G. A subgraph of G isomorphic to Z_n for some $n \geq 3$ is called a *cycle* in G. Spanning paths or cycles are commonly called *Hamiltonian*.

1.13 If $u, v \in V(G)$, a *u - v path* in G is a path in G whose endpoints are u and v. If there exists a u - v path in G, we define the *distance* $\partial(u, v)$ from u to v to be the length of the shortest u - v path in G. In particular, $\partial(u, u) = 0$. If there is no u - v path in G we define $\partial(u, v) = \infty$. If $v \in V(G)$, $U \subseteq V(G)$ we define

$\partial(v, U) = \min\{\partial(v, u) | u \in U\}$. The *diameter* of G is $\max\{\partial(u, v) | u, v \in V(G)\}$.

If $\partial(u, v)$ is finite for all $u, v \in V(G)$, then G is called *connected.* The maximal connected subgraphs of G are called its *components.* If G has no cycles it is called a *forest*; if it is also connected it is called a *tree.*

1.14 The *degree* $d_G(v)$ of a point v in a graph G is the number of edges incident with v. If v has zero degree it is called an *isolated point* of G; if it has degree one it is called an *endpoint* of G. If every point of G has the same degree, G is said to be *regular.* Generalizing the notion of degree, for any set $U \subseteq V(G)$ and $v \in V(G)$ we can define the *degree of* v *relative to* U, $d_G(v, U)$, as the number of edges incident with both v and an element of U. If it is clear from the context which graph we are referring to, the notations $d_G(v)$ and $d_G(v, U)$ can be abbreviated $d(v)$ and $d(v, U)$ respectively.

Let $G \in \underset{\sim}{G}(V)$. Then $\bar{G} \in \underset{\sim}{G}(V)$ is the *complement* of G. $\{v_1, v_2\}$ is an edge of $\bar{G}$ iff $v_1 \neq v_2$ and $\{v_1, v_2\} \notin E(G)$. The proof of the following lemma is trivial.

1.15 Lemma: *If* $G \in \underset{\sim}{G}(V)$, *then* $\Gamma(\bar{G}) = \Gamma(G)$. □

Here and elsewhere, the symbol □ indicates the end or absence of a proof.

1.16 Let $G \in \underset{\sim}{G}(V)$ where $V = \{v_1, \cdots, v_n\}$. The *adjacency matrix* of G is the $n \times n$ matrix $A = A(G)$ where $A_{ij} = 1$ if $\{v_i, v_j\} \in E(G)$ and $A_{ij} = 0$ otherwise. We use the adjacency matrix to simplify the definition of two graph operations. For any $m > 0$ define I_m to be

the $m \times m$ identity matrix and J_m to be the $m \times m$ matrix with every entry one.

Let G, H be labelled graphs, where $|V(G)| = n$, $|V(H)| = m$. The *cartesian product* $G \times H$ is defined by

$$A(G \times H) = A(G) \otimes I_m + I_n \otimes A(H).$$

The *composition* $G[H]$ is defined by

$$A(G[H]) = I_n \otimes A(H) + A(G) \otimes J_m.$$

CHAPTER TWO

LATTICES AND PARTITIONS

2.1 In this chapter we first introduce the idea of a lattice and give a few basic lemmas. We then define the lattice of partitions of a set and develop the elementary theory that will be needed later. All the results of this chapter are well-known.

2.2 Let Δ be any set. A binary relation $\leq$ on Δ is called a *partial order* if for $x, y, z \in \Delta$ we have

(i) $x \leq x$,

(ii) $x \leq y$, $y \leq x$ implies $x = y$, and

(iii) $x \leq y$, $y \leq z$ implies $x \leq z$.

If, in addition, either $x \leq y$ or $y \leq x$ for any pair of elements x, y of Δ, then $\leq$ is called a *total order* on Δ.

If $\leq$ is a partial order on Δ, then the pair $(\Delta, \leq)$ is called a *partially ordered set*, or simply *poset*. We will normally write $(\Delta, \leq)$ simply as Δ, unless it is necessary to emphasise the order relation.

Suppose Δ is a poset, and $\Delta' \subseteq \Delta$. An element $x \in \Delta$ is the *least upper bound (lub)* of Δ' if

(i) $y \leq x$ for all $y \in \Delta'$, and

(ii) if $y \leq z$ for all $y \in \Delta'$, then $x \leq z$.

Similarly, x is the *greatest lower bound (glb)* of Δ' if

(i) $y \geq x$ for all $y \in \Delta'$, and

(ii) if $y \geq z$ for all $y \in \Delta'$, then $x \geq z$.

2.3 A poset Δ is called a *lattice* if every two-element subset $\{x, y\} \subseteq \Delta$ has a glb and a lub. The glb of x and y is called their *meet* and denoted $x \wedge y$. The lub of x and y is called their *join* and denoted $x \vee y$.

Information on lattices additional to what we give here can be found in Birkhoff [7]. The following two lemmas are standard.

2.4 Lemma: [7] *Let* $(\Delta, \leq)$ *be a lattice, and let* $x, y \in \Delta$. *Then*

(i) $x \wedge x = x$, $x \vee x = x$,

(ii) $x \wedge y = y \wedge x$, $x \vee y = y \vee x$,

(iii) $x \wedge (x \vee y) = x$, $x \vee (x \wedge y) = x$. □

2.5 Lemma: *Let* $(\Delta, \leq)$ *be a lattice, where* $|\Delta|$ *is finite. Then for any subset* $\Delta' \subseteq \Delta$, *lub* Δ' *and glb* Δ' *exist.*

Proof: Let $\Delta' = \{x_1, x_2, \cdots, x_r\}$. Then

glb $\Delta' = x_1 \wedge x_2 \wedge \cdots \wedge x_r$, and

lub $\Delta' = x_1 \vee x_2 \vee \cdots \vee x_r$. □

From now on we will assume that all our lattices are finite, since we have no need for infinite ones. Sometimes lub Δ' and glb Δ' will be written $\vee(\Delta')$ and $\wedge(\Delta')$ respectively.

2.6 Lemma: *Let* $(\Delta, \leq)$ *be a lattice, and let* $\Delta' \subseteq \Delta$. *Then if* Δ' *is closed under* $\vee$ *and contains* $\wedge(\Delta)$, $(\Delta', \leq)$ *is a lattice.*

Proof: Let $x, y \in \Delta'$. By definition $x \vee y \in \Delta'$. Furthermore, the glb of x and y in Δ' can be identified as $\vee\{z \in \Delta' | z \leq x \wedge y\}$, where $\wedge$ is the meet operation in Δ. □

2.7 Let V be a finite set. A *partition* of V is a set π of disjoint non-empty subsets of V whose union is V. The elements of π are known as its *cells*. If a cell of π contains just one element $v \in V$, it will be called a *trivial cell* of π, and π will be said to *fix* v.

Two partitions of V have special names. The *discrete partition* of V consists of $|V|$ trivial cells. At the other extreme, the *unit partition* of V consists of the one cell V.

Suppose the cells of π are $C_1, C_2, \cdots, C_r$. To emphasise the fact that π is a partition we will write it as $\{C_1|C_2|\cdots|C_r\}$ rather than as $\{C_1, C_2, \cdots, C_r\}$. This will be especially convenient when actual values are given. For example, if the cells of π are $\{1, 2\}$, $\{3\}$ and $\{4, 5, 6\}$, then π will be written as $\{1, 2|3|4, 5, 6\}$ or simply as $\{1, 2|4, 5, 6\}$, in which case elements of V not mentioned are assumed to be in trivial cells.

2.8 The collection of all partitions of V will be denoted $\Pi(V)$. We now proceed to define a partial order $\leq$ on $\Pi(V)$ and then to show that $(\Pi(V), \leq)$ is a lattice.

Let $\pi_1, \pi_2 \in \Pi(V)$. We say that π_1 is *finer* than π_2, written $\pi_1 \leq \pi_2$ if for every cell $C_1 \in \pi_1$ there exists a cell $C_2 \in \pi_2$

such that $C_1 \subseteq C_2$. In the same circumstances we call π_2 *coarser* than π_1 ($\pi_2 \geq \pi_1$). For example, $\{1, 2|3|4, 5\} \leq \{1, 2, 3|4, 5\}$.

2.9 Lemma: *$\leq$ is a partial order on $\Pi(V)$.*

Proof: Referring to the definition of partial order (2.2) we see that conditions (i) and (ii) are satisfied trivially. Condition (iii) follows from the transitivity of set inclusion. □

2.10 Lemma: *Suppose $\pi_1 \leq \pi_2$ where $\pi_1, \pi_2 \in \Pi(V)$. Then each cell of π_2 is a union of cells of π_1.*

Proof: If the lemma is not true, there are cells $C_1 \in \pi_1$ and $C_2 \in \pi_2$ such that both $C_1 \cap C_2$ and $C_1 \backslash C_2$ are not null. But then π_1 is not finer than π_2. □

2.11 Lemma: *Suppose $\pi_1, \pi_2 \in \Pi(V)$. Then glb $\{\pi_1, \pi_2\}$ exists (under $\leq$).*

Proof: Define $\pi = \{C \neq \phi | C = C_1 \cap C_2, C_1 \in \pi_1, C_2 \in \pi_2\}$. Clearly $\pi \in \Pi(V)$ and $\pi \leq \pi_1$, $\pi \leq \pi_2$.

Now suppose that for some $\pi' \in \Pi(V)$, we have $\pi' \leq \pi_1$ and $\pi' \leq \pi_2$. Then for any $C' \in \pi'$, there are cells $C_1 \in \pi_1$ and $C_2 \in \pi_2$ such that $C' \subseteq C_1$ and $C' \subseteq C_2$. Consequently, $C' \subseteq C_1 \cap C_2$ and so $\pi' \leq \pi$.

Therefore, $\pi = \text{glb } \{\pi_1, \pi_2\}$. □

2.12 Lemma: *Suppose $\pi_1, \pi_2 \in \Pi(V)$. Then lub $\{\pi_1, \pi_2\}$ exists.*

Proof: Define a graph $G \in \underset{\sim}{G}(V)$ as follows. Two points $v_1, v_2 \in V$ are connected iff $v_1 \neq v_2$ and v_1 and v_2 are either in the same cell of π_1 or in the same cell of π_2.

Let $\pi \in \Pi(V)$ be the partition whose cells correspond to the components of G. Trivially, $\pi_1 \leq \pi$ and $\pi_2 \leq \pi$.

Now suppose that for some $\pi' \in \Pi(V)$ we have $\pi_1 \leq \pi'$ and $\pi_2 \leq \pi'$. Let $C \in \pi$ and $v_1, v_2 \in C$. Then there is a path in G of the form $v_1 = w_0, w_1, \cdots, w_r = v_2$.

If w_0 is in cell C' of π', then either w_0 and w_1 are in the same cell of π_1, in which case $\pi_1 \leq \pi'$ implies $w_1 \in C'$, or in the same cell of π_2, in which case $\pi_2 \leq \pi'$ implies $w_1 \in C'$. Continuing along the path in this fashion we see that v_1 and v_2 are both in C', and so $C \subseteq C'$.

Therefore $\pi \leq \pi'$ and so $\pi = \text{lub } \{\pi_1, \pi_2\}$. □

From 2.9, 2.11 and 2.12 we have the following result.

2.13 Theorem: $(\Pi(V), \leq)$ *is a lattice.* □

If $\pi_1, \pi_2 \in \Pi(V)$, then the notations $\pi_1 \wedge \pi_2$ and $\pi_1 \vee \pi_2$ will always indicate the meet and join *in the lattice* $(\Pi(V), \leq)$ even though we consider other lattices of partitions.

2.14 There is a natural correspondence between $\Pi(V)$ and the family of equivalence relations on V. Given $\pi \in \Pi(V)$, we can define the equivalence $v_1 \underset{\pi}{\sim} v_2$ iff v_1 and v_2 are in the same cell of π. Conversely, given an equivalence relation defined on V, we can find

a partition whose cells are the equivalence classes.

We conclude with a final point of notation. Suppose $\pi \in \Pi(V)$ and that $U \subseteq V$ is a union of cells of π. Then $\pi|_U$ is the partition of U whose cells are those cells of π contained in U. $\pi|_U$ might be called the partition of U *induced by* π. For example, if $\pi = \{1, 2|3|4, 5, 6\}$ and $U = \{1, 2, 3\}$, then $\pi|_U = \{1, 2|3\}$.

CHAPTER THREE

TECHNICAL PREREQUISITES

3.1 In this chapter we present various items of technical information which will be necessary for the proper evaluation of the material in later chapters. We begin by describing the computer on which our algorithms were implemented, and discuss the methods used for representing various data items. Also in this chapter we give an algorithm for computing a generalized form of the join of two partitions. Finally, the concept of a "random graph" is discussed.

3.2 The algorithms described in this thesis have been implemented on a CDC Cyber 70 model 73. The languages used have been FORTRAN and assembly language (COMPASS).

In order that the execution times we present may be approximately translated into the context of another machine, we list a few of the basic operations and their execution times in microseconds.

FETCH	1·2
STORE	1·0
BOOLEAN OPERATION	0·5
SHIFT (any length)	0·6
POPULATION COUNT	6·8

The population count instruction counts the number of one-bits in a word, and has proved especially useful. The Cyber has a word size of 60 bits. Consequently our implementations have been restricted to graphs of from 1 to 60 points, although larger graphs may be accommodated with more complicated programming.

3.3 In the description of algorithms in this thesis we have attempted to adopt a free and simple format without sacrificing rigour, but without adhering to any formal code. Briefly,

(1) The operator := indicates an assignment of value as in ALGOL. For example the statement

$$i := i + 2$$

means "increment i by 2", thus avoiding the contradiction $i = i + 2$.

(2) Recursive definitions will be avoided (even if occasionally at the expense of elegance).

(3) Semicolons will be used freely for punctuation. They do not have any special significance.

(4) Unless otherwise specified, control flows from one step to the next. The statement "stop" terminates execution.

As an example we give an algorithm for a generalized form of the join operation introduced in Section 2.12. We shall need this algorithm in Chapter Seven.

3.4 Let V be a finite set, and let $V_1 \subseteq V$, $V_2 \subseteq V$. Take partitions $\pi_1 \in \Pi(V_1)$, $\pi_2 \in \Pi(V_2)$.

We define a graph G as follows. The points of G are the elements of V_1. If $v_1, v_2 \in V_1$, then v_1 and v_2 are connected iff

$$v_1 \underset{\pi_1}{\sim} v_2 \quad \text{or} \quad v_1 \underset{\pi_2}{\sim} v_2.$$

$\pi_1 \bar{\vee} \pi_2$ can now be defined as the partition in $\Pi(V_1)$ whose cells correspond to the components of G.

3.5 Lemma:

(1) *If* $V_1 \cap V_2 = \phi$, *then* $\pi_1 \bar{\vee} \pi_2 = \pi_1$.

(2) *In general* $\pi_1 \bar{\vee} \pi_2 \neq \pi_2 \bar{\vee} \pi_1$. *However, if* $V_1 = V_2$, *then* $\pi_1 \bar{\vee} \pi_2 = \pi_2 \bar{\vee} \pi_1 = \pi_1 \vee \pi_2$. □

We now give an algorithm for finding $\pi = \pi_1 \bar{\vee} \pi_2$. Suppose $\pi_2 = \{D_1|D_2|\cdots|D_\ell\}$ $(1 \leq \ell)$.

3.6 Algorithm: Compute $\pi = \pi_1 \bar{\vee} \pi_2$.

(1) Set $\pi := \pi_1$; $i := 1$. Suppose π is the partition $\{C_1|C_2|\cdots|C_r\}$.

(2) Set $k := 1$.

(3) If $k \geq r$ go to step (9).

(4) If $D_i \cap C_k \neq \phi$, go to step (5). Otherwise set $k := k + 1$ and go to step (3).

(5) Set $j := k + 1$; $j' := k$.

(6) If $C_j \cap D_i = \phi$ set $j' := j' + 1$ and $C_{j'} := C_j$. Otherwise set $C_k := C_k \cup C_j$.

(7) Set $j := j + 1$. If $j \leq k$ go to step (6).

(8) Set $r := j'$.

(9) Set $i := i + 1$. If $i \leq \ell$ go to step (2). Otherwise stop.

3.7 Theorem: *At the termination of Algorithm 3.6,* $\pi = \{C_1|C_2|\cdots|C_r\}$ *is the generalized join* $\pi_1 \bar{\vee} \pi_2$.

Proof:

(1) Note that at step (6) we always have $j' \leq j$ and so the assignment $C_{j'} := C_j$ does not destroy cells which have not been examined.

(2) The effect of steps (5)-(7) is to merge all those cells

of π which have non-zero intersection with D_i. Since no other changes are made to π (as an unordered collection of cells), we must have $\pi \leq \pi_1 \bar{\vee} \pi_2$.

(3) Suppose v_1 and v_2 are in the same class of $\pi_1 \bar{\vee} \pi_2$. Then there exists (by definition) a sequence of points $v_1 = w_0, w_1, \cdots, w_k = v_2$ of V_1 so that for $1 \leq i \leq k$,

$$w_{i-1} \tilde{\pi}_1 w_i \quad \text{or} \quad w_{i-1} \tilde{\pi}_2 w_i.$$

Suppose that for some i, w_{i-1} and w_i are not in the same cell of π_1. Then there exists a cell D of π_2 so that w_{i-1} and w_i are in D. However, the cells of π containing w_{i-1} and w_i will be merged when D is being considered in steps (4)-(7) of the algorithm. Hence $\pi \geq \pi_1 \bar{\vee} \pi_2$. □

3.8 The efficiency of most graph theoretic algorithms, including those presented here, is highly dependent on the way in which the data items are stored in the computer. In our case the data items to be considered are graphs and partitions.

3.9 The two most common forms in which a graph (or digraph) can be stored in a computer are the adjacency matrix and the adjacency list.

In the latter method, each point is associated with a list of those points adjacent to it. In this form questions like "What is the next point adjacent to v?" are very easily answered. This type of representation is especially useful when the number of edges is small as, for example, in planar graphs.

In the former method each of the n^2 entries of the adjacency

matrix is stored. Since each entry is either 0 or 1, it requires only one bit. The usual system, and the one we employ here, is to store each row of the adjacency matrix in a separate machine word (assuming that n is not too large). This has several advantages:

(1) Only n words are required to store the entire matrix.

(2) Set operations between the rows (AND, OR etc.) can be performed in single machine operations, thus achieving a degree of parallelism.

(3) The position of one-bits in a word can be found by use of the floating-point normalisation instruction on most machines. This involves adding an exponent to all or part of a word and then observing the new exponent after normalisation. It does not involve a bit-by-bit search of the word as is often assumed.

Further discussion and references can be found in Corneil [13] and Kirkpatrick [32]. Another means of representing a graph, the "K-formula" has been studied by Krider [35] and by Berztiss [5].

3.10 The following storage method for partitions has been found convenient.

Let $\pi = \{C_1|C_2|\cdots|C_k\}$. Then π is represented by k machine words $w_1, \cdots, w_k$ where bit i of word j ($1 \leq i \leq n$, $1 \leq j \leq k$) is set to one iff $i \in C_j$.

This form of representation was chosen to simplify partition operations and for compatibility with the structure used for graphs.

In some circumstances it is convenient to keep track of

those cells which have exactly one element. This can be done by keeping an extra machine word whose one-bits indicate these cells.

3.11 Once a graph-theoretic algorithm has been implemented there are several approaches which can be made towards its practical evaluation.

In one approach the performance of the algorithm is examined when it is applied to a specifically selected class of graphs. For example, it can be applied to all the graphs on a small number of points or to members of recognised families (paths, cycles etc.). Alternatively, graphs may be constructed in an attempt to bring out the worst of an algorithm, in order to guess at its "worst-case" behaviour.

A fundamentally different approach is to apply the algorithm to a collection of graphs chosen in some "random" manner from a larger class. For example Kühn [36], [37] has devised a procedure by which "random" graphs having a specified degree sequence can be constructed.

For our own purposes we have found the following process convenient. Let $0 \leq \sigma \leq 1$ be a real number, and let $n \geq 1$. Suppose the edges of the complete graph are labelled $e_1, e_2, \cdots, e_N$ where $N = \binom{n}{2}$. Then we can construct a graph G on n points as follows. For each $1 \leq i \leq N$ generate a random number x from a population rectangularly distributed between 0 and 1. If $x \leq \sigma$ then we include the edge e_i in G; otherwise we leave it out.

A sequence of graphs produced in this manner for say $\sigma = 0{\cdot}5$ will be referred to as "random graphs with $\sigma = 0{\cdot}5$". The numbers of

edges in the graphs of the sequence clearly have a binomial distribution with mean $N\sigma$ and variance $N\sigma(1 - \sigma)$. For a fixed number of edges m every labelled graph with n points and m edges has an equal probability of occurring.

CHAPTER FOUR

PERMUTATION GROUPS

4.1 In this chapter we consider a few basic results on permutation groups. Sections 4.2 to 4.6 are standard and can be found in any reasonable text. The results in Sections 4.7 to 4.11 are unlikely to be new, but we have not seen them previously in print. In the last part of the chapter we consider a lattice $\Theta(\Psi)$ defined by the orbits of subgroups of Ψ. While this lattice seems to have been rarely defined, the results we obtain about it are well known.

Let Ψ be a group of permutations of the points V, where $V = \{1, 2, 3, \cdots, n\}$.

Let $v_1 \in V$ and suppose $v_1^{\Psi} = \{v_1, v_2, \cdots, v_s\}$.

4.2 Lemma: *Ψ can be written as the disjoint union*

$$\Psi_{v_1}\gamma_1 \cup \Psi_{v_1}\gamma_2 \cup \cdots \cup \Psi_{v_1}\gamma_s,$$

where Ψ_{v_1} is the stabiliser of v_1 in Ψ and for $1 \leq i \leq s$, γ_i is any element of Ψ such that $v_1^{\gamma_i} = v_i$.

Proof: Clearly $\Psi_{v_1}\gamma_1 \cup \Psi_{v_1}\gamma_2 \cup \cdots \cup \Psi_{v_1}\gamma_s$ is contained in Ψ. If $i \neq j$, then $\Psi_{v_1}\gamma_i \cap \Psi_{v_1}\gamma_j = \phi$ since elements of $\Psi_{v_1}\gamma_i$ take v_1 onto v_i whereas elements of $\Psi_{v_1}\gamma_j$ take v_1 onto v_j.

Let $\gamma \in \Psi$. Then $v_1^{\gamma} = v_i$ for some i. Therefore $v_1^{\gamma\gamma_i^{-1}} = v_i^{\gamma_i^{-1}} = v_1$.

Hence $\gamma\gamma_i^{-1} \in \Psi_{v_1}$ so that $\gamma \in \Psi_{v_1}\gamma_i$. □

4.3 Corollary: (Orbit-stabiliser relation) $|\Psi| = |\Psi_{v_1}||v_1{}^{\Psi}|$. □

The sets $\Psi_{v_1}\gamma_i$ $(1 \le i \le s)$ are called the (right) *cosets of* Ψ_{v_1} *in* Ψ and the set $\{\gamma_i\}_1^s$ is a set of (right) *coset representatives for* Ψ_{v_1} *in* Ψ.

Let $\{v_1, \cdots, v_{r+1}\} \subseteq V$ be a set of points such that the point-wise stabiliser $\Psi_{v_1,\cdots,v_{r+1}}$ is trivial. For $1 \le k \le r+1$ denote $\Psi_{v_1,\cdots,v_k}$ by $\Psi^{(k)}$ and Ψ by $\Psi^{(0)}$.

In the manner of 4.2 write

4.4 $$\Psi^{(k)} = \bigcup_{i=1}^{s_k} \Psi^{(k+1)}\gamma_i^{(k)}, \text{ where } s_k = |v_{k+1}{}^{\Psi^{(k)}}| \quad (0 \le k \le r).$$

4.5 Lemma: *For* $0 \le h \le r$, $\Psi^{(h)}$ *is generated by the set* $\Omega_h = \{\gamma_i^{(k)} | h \le k \le r, 1 \le i \le s_k\}$.

Proof: The lemma is clearly true when $h = r$. Suppose it is true when $h = j$ where $1 \le j \le r$. In other words, suppose $\langle \Omega_j \rangle = \Psi^{(j)}$.

Then by 4.2 and 4.4, any element of $\Psi^{(j-1)}$ can be written (uniquely) in the form $\gamma\gamma_i^{(j-1)}$, where $\gamma \in \Psi^{(j)}$ and $\gamma_i^{(j-1)} \in \Omega_{j-1}$.

Hence Ω_{j-1} generates $\Psi^{(j-1)}$. □

4.6 Lemma: $$|\Psi| = \prod_{k=0}^{r} s_k.$$

Proof: From 4.3 and 4.4. □

4.7 Theorem: *Let* Y *be a subset of* Ψ. *For* $0 \le k \le r+1$ *define* $Y_k = Y \cap \Psi^{(k)}$. *Then if the orbit of* $\langle Y_k \rangle$ *containing* v_{k+1} *is the same*

as the orbit of $\Psi^{(k)}$ *containing* v_{k+1} *for* $k \geq h \geq 0$ *we have*

$$\langle Y_h \rangle = \Psi^{(h)}.$$

In particular, if $h = 0$, $\langle Y \rangle = \Psi$.

Proof: The theorem is trivially true for $h = r + 1$.

Suppose it is true for $1 \leq h = \ell + 1 \leq r + 1$.

Obviously $\langle Y_\ell \rangle \leq \Psi^{(\ell)}$. Let $\{w_1, \cdots, w_s\}$ be the orbit of $\Psi^{(\ell)}$ containing $v_{\ell+1}$. By 4.2, $\Psi^{(\ell)} = \Psi^{(\ell+1)}{}_{\gamma_1} \cup \cdots \cup \Psi^{(\ell+1)}{}_{\gamma_s}$ where for $1 \leq i \leq s$, γ_i is any element of $\Psi^{(\ell)}$ such that $w_1^{\gamma_i} = w_i$.

But $\{w_1, \cdots, w_s\}$ is an orbit of $\langle Y_\ell \rangle$ by hypothesis and so such a set $\{\gamma_1, \cdots, \gamma_s\}$ can be found in $\langle Y_\ell \rangle$.

Hence $\langle Y_\ell \rangle = \Psi^{(\ell)}$. □

4.8 Theorem: *Let* $Y \subseteq \Psi$ *satisfy the requirements of 4.7 for* $h = 0$. *Then* Y *has a subset* Y' *satisfying these requirements and such that* $|Y'| \leq n - p$ *where* Ψ *has* p *orbits.*

Proof: Label the elements of Y as $\gamma_1, \cdots, \gamma_t$ in an order such that if $\gamma_i \notin \Psi^{(k)}$ and $\gamma_j \in \Psi^{(k)}$ for some i, j, k then $i < j$.

Then the required set Y' can be produced by the following algorithm.

4.9 Algorithm: Compute the generators Y' for Ψ.

(1) Set $Y' := \phi$;

$\pi :=$ discrete partition of V;

$i := 1$.

(2) Set π' to the partition of V whose cells correspond to the cycles of γ_i. If π' is not finer than π set $\pi := \pi' \vee \pi$ and $Y' := Y' \cup \{\gamma_i\}$.

(3) Set $i := i + 1$. If $i \leq t$ go to step (2); otherwise stop.

The element γ_i is not accepted into Y' if, and only if, $\langle \gamma_i, \gamma_{i+1}, \cdots, \gamma_t \rangle$ has the same orbits as $\langle \gamma_{i+1}, \cdots, \gamma_t \rangle$. Hence, by the ordering of Y, $\langle Y' \cap \Psi^{(k)} \rangle$ has the same orbits as $\langle Y \cap \Psi^{(k)} \rangle$ for any k. Therefore the set Y' satisfies the requirements of Theorem 4.7 for $h = 0$.

Now the partition π has n cells at the start of the algorithm and p at the finish. Furthermore, the number of cells is decreased each time an element is added to Y'.

Therefore $|Y'| \leq n - p$. □

4.10 Note: The set Ω_0 defined in 4.5 satisfies the requirements of Theorem 4.7. Therefore by 4.8 it contains a subset of at most $n - p$ elements which generate Ψ.

4.11 Given a subset Y of Ψ which satisfies the requirements of 4.7 for the sequence $v_1, v_2, \cdots, v_{r+1}$ it is a straightforward matter to generate the whole of Ψ. The first step requires the construction of coset representatives $\{\gamma_i^{(k)}\}$ satisfying 4.4. This can be done (for each k) by defining a digraph G as follows. The points of G are the elements of V. The edges of G are the directed pairs $[v, v^\gamma]$ where $v \in V$, $\gamma \in Y \cap \Psi^{(k)}$ and $v^\gamma \neq v$. The directed edge $[v, v^\gamma]$ is labelled with the element γ, with the proviso that no directed edge need be labelled with more than one group element. Now let $\{w_1, \cdots, w_s\}$ be

the component of G containing $w_1 = v_{k+1}$. For each w_i $(1 \le i \le s)$ there is a path of directed edges from w_1 to w_i. Then define $\gamma_i^{(k)} = \delta_1\delta_2\cdots\delta_\ell$ where $\delta_1, \cdots, \delta_\ell$ are the labels of the edges of the path chosen.

Clearly the above procedure generates coset representatives $\{\gamma_i^{(k)}\}$ satisfying 4.4. Once this has been done the generation of Ψ is routine. Every element of Ψ can obviously be written in the form $\gamma_{i_1}^{(1)}\gamma_{i_2}^{(2)}\cdots\gamma_{i_r}^{(r)}$, and by 4.6 this decomposition is unique. Although we will not give further details here an algorithm based on these ideas can be devised which for large $|\Psi|$ requires only marginally more than one permutation multiplication for each element of Ψ.

Let γ be an element of Ψ. Then γ is said to *fix* a partition $\pi \in \Pi(V)$ if $v \underset{\pi}{\sim} v^\gamma$ for all $v \in V$. The set of all elements of Ψ which fix π is denoted by Ψ_π, and is called the *stabiliser of* π *in* Ψ.

For example, if $n = 4$, $\Psi = S_4$ and $\pi = \{1,2|3,4\}$ then Ψ_π is the set $\{(1), (1\ 2), (3\ 4), (1\ 2)(3\ 4)\}$.

4.12 Lemma: *Let* $\pi_1, \pi_2 \in \Pi(V)$. *Then* $\Psi_{\pi_1} \le \Psi_{\pi_2}$ *iff* $\pi_1 \le \pi_2$. □

4.13 Corollary: $\Psi_\pi \le \Psi$ *for any* $\pi \in \Pi(V)$.

Proof: $\Psi = \Psi_{\pi_0}$ where π_0 is the unit partition of V. □

Let X be a subset of Ψ. Then we denote by θ_X the partition whose cells correspond to the orbits of the group generated by X.

In particular, if $\gamma \in \Psi$ then the cells of $\theta_{\{\gamma\}}$, which we

write as θ_γ, correspond to the cycles of γ. If X is null θ_X will be taken to be the discrete partition of V.

4.14 Theorem: *If* $X_1, X_2 \subseteq \Psi$ *then* $\theta_{X_1 \cup X_2} = \theta_{X_1} \vee \theta_{X_2}$.

Proof: Let v_1 and v_2 be in the same cell of $\theta_{X_1 \cup X_2}$. Then there is a sequence $\gamma_1, \gamma_2, \cdots, \gamma_r$ of elements of $X_1 \cup X_2$, such that $v_2 = v_1{}^\gamma$ where $\gamma = \gamma_1\gamma_2\cdots\gamma_r$. From this we can construct a sequence $w_0, w_1, \cdots, w_r$ of points by setting $w_0 = v_1$, $w_i = w_{i-1}{}^{\gamma_i}$ $(1 \leq i \leq r)$.

For $1 \leq i \leq r$ we see that w_{i-1} and w_i are either in the same cell of θ_{X_1} or of θ_{X_2} and so v_1 and v_2 are in the same cell of $\theta_{X_1} \vee \theta_{X_2}$.

The converse follows in a similar fashion. □

4.15 We can now define

$$\Theta(\Psi) = \{\pi \in \Pi(V) \mid \pi = \theta_X \text{ for some } X \subseteq \Psi\}.$$

$\Theta(\Psi)$ will be called the *orbits lattice of* Ψ and its elements will be called *orbital* with respect to Ψ.

$\Theta(\Psi)$ is not in general isomorphic to the lattice of subgroups of Ψ since distinct subgroups may have the same orbits.

Define a function θ by $\theta(\pi) = \theta_{\Psi_\pi}$ for any $\pi \in \Pi(V)$. $\theta(\pi)$ is thus the coarsest orbital partition which is finer than π. Note that θ depends on Ψ.

Then we can equivalently define $\Theta(\Psi)$ as

4.16 $$\Theta(\Psi) = \{\pi \in \Pi(V) \mid \pi = \theta(\pi)\}.$$

4.17 Example: Consider the group

$$\Psi = \{(1), (1\ 2), (7\ 8), (1\ 2)(7\ 8), (1\ 7)(2\ 8)(3\ 6)(4\ 5),$$
$$(1\ 8\ 2\ 7)(3\ 6)(4\ 5), (1\ 7\ 2\ 8)(3\ 6)(4\ 5), (1\ 8)(2\ 7)(3\ 6)(4\ 5)\}.$$

Then $\theta(\Psi)$ is the lattice of Figure 4.1.

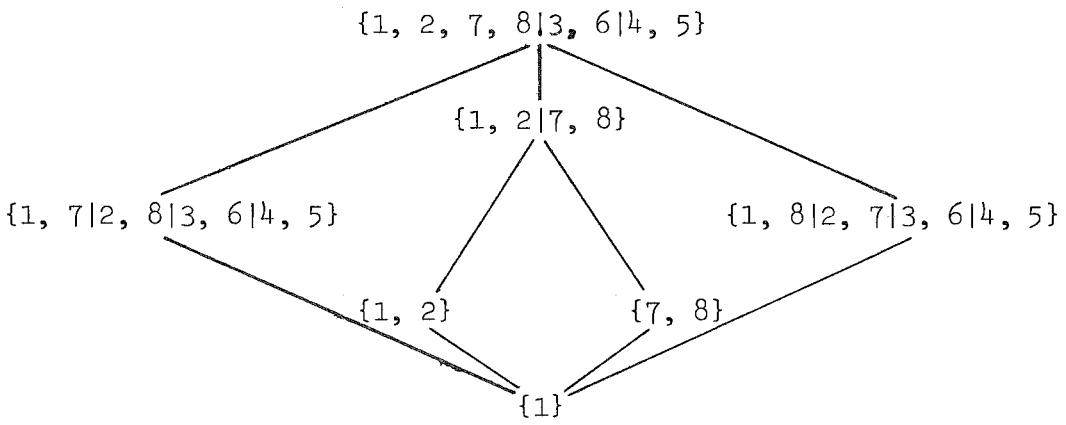


Figure 4.1

4.18 Lemma: *$\theta(\Psi)$ is closed under $\vee$ but is not necessarily closed under $\wedge$.*

Proof: The first part is immediate from 4.14. For the second part consider the partitions $\pi_1 = \{1, 7|2, 8|3, 6|4, 5\}$ and $\pi_2 = \{1, 2|7, 8\}$ of Figure 4.1. Then $\pi_1 \wedge \pi_2 = \{3, 6|4, 5\}$ which is not in the lattice. □

4.19 Lemma: *If $\pi_1, \pi_2 \in \theta(\Psi)$ then the meet of π_1 and π_2 in the lattice $\theta(\Psi)$ is $\theta(\pi_1 \wedge \pi_2)$.*

Proof: From 2.6, noting that the discrete partition of V is always in $\theta(\Psi)$. □

CHAPTER FIVE

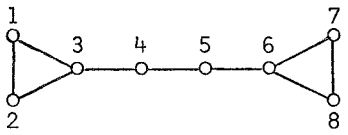
EQUITABLE PARTITIONS

5.1 In this chapter we examine the lattice of equitable partitions of the points of a graph. This lattice, although it is rarely defined as such, plays a central role in many existing algorithms for graph isomorphism [14, 46, 51, 62, 69] and in our own. Results not attributed to other authors are either new or trivial. Later in the chapter we present a new algorithm for computing the coarsest equitable partition finer than a given partition, an operation related to Unger's "extending" process [76]. We show that it is at least one order of n faster than the usual algorithm.

5.2 Until otherwise specified, G is a graph with points $V = \{1, 2, \cdots, n\}$ and $\Gamma = \Gamma(G)$ is its automorphism group. $\Theta(\Gamma)$ is the orbits lattice of Γ defined in 4.15.

Let $\pi \in \Pi(V)$ and $C_1, C_2 \in \pi$. Then C_1 is said to be *equitably joined* to C_2 (in G) if $d(v, C_2)$ is constant for all $v \in C_1$. If any pair of cells of π (not necessarily distinct) are equitably joined to each other then, following Schwenk [63], we say that π is *equitable*. The set of all equitable partitions for G will be called the *equitable partitions lattice* of G and denoted by $\Xi(G)$.

Consider for example the graph drawn in Figure 5.1. The lattice $\Xi(G)$ consists of the eight partitions illustrated.



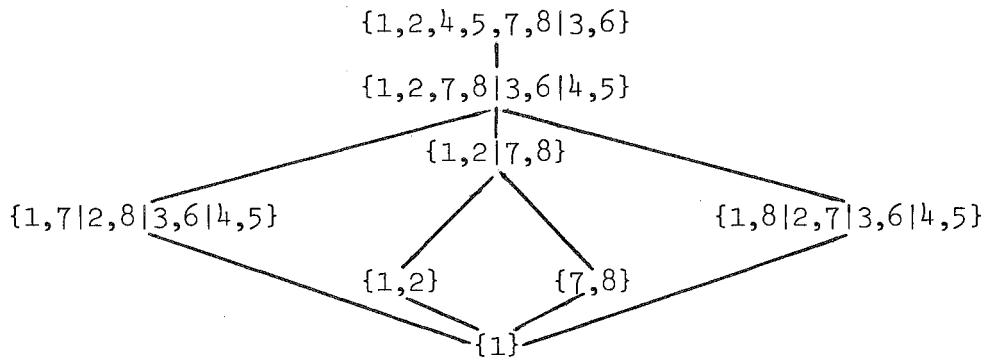


Figure 5.1

A Graph and its Equitable Partitions Lattice

The following result justifies, by 2.6 , our referring to $\Xi(G)$ as a lattice.

5.3 Lemma: $\Xi(G)$ *is closed under* $\vee$ *but not necessarily under* $\wedge$.

Proof: Let $\pi_1, \pi_2 \in \Xi(G)$ and let $\pi = \pi_1 \vee \pi_2$, $C \in \pi$.

Since $\pi_1 \leq \pi$, C is a union of cells of π_1 (2.10). Hence if $v_1 \underset{\pi_1}{\sim} v_2$, then

$$d(v_1, C) = d(v_2, C),$$

since

$$d(v_1, C_1) = d(v_2, C_1)$$

for each $C_1 \in \pi_1$.

Similarly,

$$d(v_1, C) = d(v_2, C)$$

if $v_1 \underset{\pi_2}{\sim} v_2$.

Now suppose that $v_1 \underset{\pi}{\sim} v_2$. Then as in the proof of 2.12 there is a sequence of points

$$v_1 = w_0, w_1, w_2, \cdots, w_r = v_2$$

with the property that whenever $1 \leq i \leq r$, either $w_{i-1} \underset{\pi_1}{\sim} w_i$ or $w_{i-1} \underset{\pi_2}{\sim} w_i$.

Hence $d(w_{i-1}, C) = d(w_i, C)$ for $1 \leq i \leq r$, and so $d(v_1, C) = d(v_2, C)$.

Thus $\pi \in \Xi(G)$.

To demonstrate the second part of the lemma, consider the partitions $\pi_1 = \{1, 7|2, 8|3, 6|4, 5\}$, $\pi_2 = \{1, 8|2, 7|3, 6|4, 5\}$ of Figure 5.1. Then $\pi_1 \wedge \pi_2 = \{3, 6|4, 5\}$, which is not equitable. □

It would be interesting to characterise those graphs for which $\Xi(G)$ is closed under $\wedge$, but nothing seems to be known about this problem.

5.4 For any $\pi \in \Pi(V)$ we define $\xi(\pi)$ to be the coarsest equitable partition (with respect to G) which is finer than π. For example, if G is the graph of Figure 5.1 and $\pi = \{1, 2, 7, 8|3, 6, 4, 5\}$ then $\xi(\pi) = \{1, 2, 7, 8|3, 6|4, 5\}$.

Formally, $\xi(\pi) = \vee\{\pi' \in \Xi(G) | \pi' \leq \pi\}$. The set here is not null since the discrete partition is always equitable. The join exists by 2.5 and is equitable by 5.3.

5.5 The meet of two partitions π_1 and π_2 in the lattice $\Xi(G)$ can now be identified as $\xi(\pi_1 \wedge \pi_2)$.

The following results (5.6-5.10) are elementary and well known.

5.6 Lemma: *Let* $\pi \in \Xi(G)$ *and* $\pi' \subseteq \pi$. *If* G' *is the subgraph of* G *induced by the points in* π', *then* $\pi' \in \Xi(G')$.

Proof: If $C \in \pi'$ and $v_1 \underset{\pi_1}{\sim} v_2$, then $d_{G'}(v_1, C) = d_G(v_1, C) = d_G(v_2, C) = d_{G'}(v_2, C)$. □

5.7 Corollary: *The subgraph of* G *induced by the points in one cell of* $\pi \in \Xi(G)$ *is regular.* □

5.8 Lemma: $\Xi(\bar{G}) = \Xi(G)$.

Proof: Let $\pi \in \Xi(G)$, $C \in \pi$ and $v \in V$. Then

$$d_{\bar{G}}(v, C) = \begin{cases} |C| - d_G(v, C) & \text{if} \quad v \in C \\ |C| - d_G(v, C) - 1 & \text{if} \quad v \notin C. \end{cases}$$

Hence if $v_1 \underset{\pi}{\sim} v_2$ then $d_{\bar{G}}(v_1, C) = d_{\bar{G}}(v_2, C)$. Therefore $\Xi(G) \subseteq \Xi(\bar{G})$ and the opposite inequality follows similarly. □

5.9 Theorem: $\Theta(\Gamma) \subseteq \Xi(G)$.

Proof: Let $\pi \in \Theta(\Gamma)$. Then $\pi = \theta(\pi)$.

Let C be a cell of π and let $v_1 \underset{\pi}{\sim} v_2$. Then there is an element γ of Γ_π such that $v_1^{\gamma} = v_2$.

Now γ maps C onto itself, and v_1 is joined to $v \in C$ iff $v_2 = v_1^{\gamma}$ is joined to v^{γ}.

Hence $d(v_1, C) = d(v_2, C)$ and so $\pi \in \Xi(G)$. □

5.10 Corollary: *For any* $\pi \in \Pi(V)$, $\theta(\pi) \leq \xi(\pi)$. □

The conclusion of the last theorem suggests the following definition. A graph G will be called *simply-equitable* (or s-e for brevity) if equality holds in 5.9. That is to say, G is s-e if $\theta(\Gamma) = \Xi(G)$. The characterization of s-e graphs appears to be very difficult and only partial results have been obtained.

The smallest graphs which are not s-e are the disconnected graph

and its complement. In these cases the unit partition is equitable but not orbital.

In practice it is very difficult to tell whether a given graph is s-e or not, due to the large size of $\Pi(V)$ for moderate n. However, if it is s-e, then the coarsest equitable partition is also the coarsest orbital partition. That is,

5.11 $$\theta_{\Gamma} = v(\Xi(G)).$$

This necessary (but not sufficient) condition is readily tested empirically. A search of all the graphs with 8 points has shown that 5.11 holds except for those shown in Figure 5.2, together with

their complements. The graph marked (*) is self-complementary. The coarsest equitable partition is indicated by the labelling; two points are in the same cell if they have the same label (or no label). It is not known whether there are any 8-point graphs satisfying 5.11 but not s-e.

A similar search of the 274668 graphs with 9 points has revealed 168 for which 5.11 does not hold.

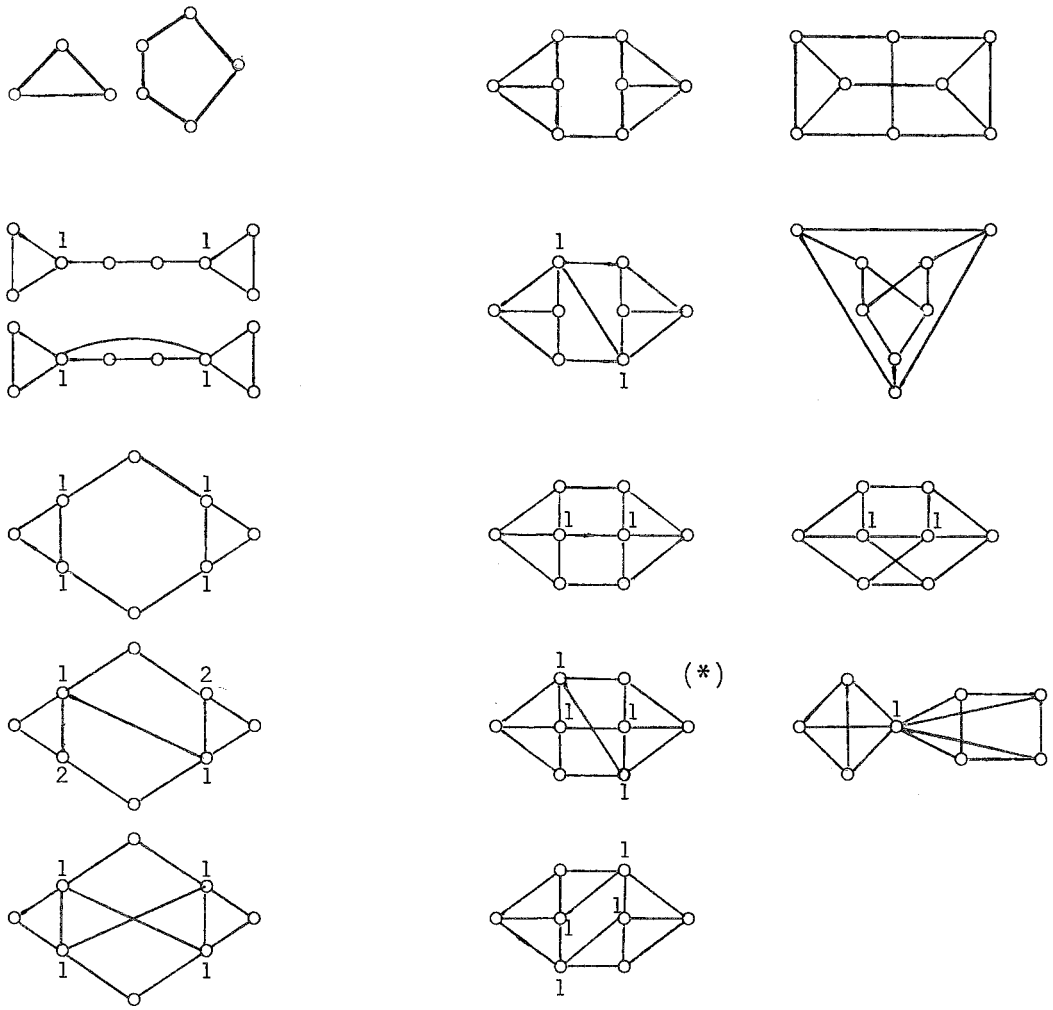
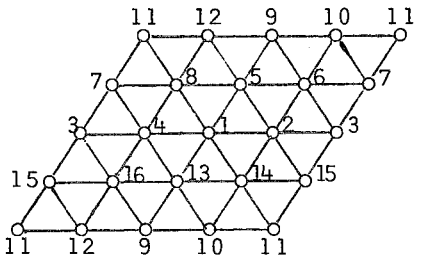


Figure 5.2

8-Point Graphs not Satisfying 5.11

It is not easy to find graphs which satisfy 5.11 but are not s-e. In fact it seems to be usually the case that all equitable partitions of G which are not orbital are coarser than θ_Γ. The smallest

counter-example known to the author is the 16-point graph shown in Figure 5.3, which appeared in [66] in a different context. This graph is transitive and so 5.11 is satisfied. However, if $\pi = \{1|2, 4, 5, 8, 13, 14|3, 6, 7, 9, 10, 11, 12, 15, 16\}$ then π is equitable, but $\theta(\pi) = \{1|2, 4, 5, 8, 13, 14|3, 9, 11|6, 7, 10, 12, 15, 16\}$.



Points with the same label are to be identified.

Figure 5.3

Corneil [11] has proved that all trees satisfy 5.11. We can generalize this result considerably as follows.

If G is tree define $\underline{\underline{G}} = K_1$. If G is not a tree but is connected, define $\underline{\underline{G}}$ to be the largest induced subgraph of G which has no points of degree zero or one.

5.12 Lemma: *If G is connected, then $\underline{\underline{G}}$ is uniquely defined and connected.*

Proof: If G is a tree the lemma is trivial.

Suppose G is not a tree and G_1 and G_2 are different induced subgraphs satisfying the definition of $\underline{\underline{G}}$. Then let G_3 be the subgraph of G induced by the points of G_1, the points of G_2 and the points of every path in G joining a point of G_1 to a point of G_2. Then G_3 has no points of degree zero or one and is larger than either G_1 or G_2.

If $\underline{\underline{G}}$ is disconnected, then by including the points of

every path joining one component of $\underline{\underline{G}}$ to another we derive a similar contradiction. □

If G is disconnected and has components $G_1, G_2, \cdots, G_r$ then we define $\underline{\underline{G}}$ to be the disjoint union $\underline{\underline{G}}_1 \cup \underline{\underline{G}}_2 \cup \cdots \cup \underline{\underline{G}}_r$.

For example the graph G of Figure 5.4 has $\underline{\underline{G}}$ as shown.

G: 

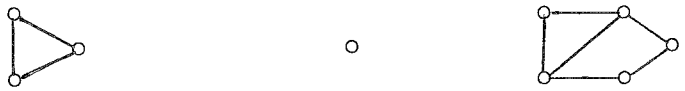
$\underline{\underline{G}}$: 

Figure 5.4

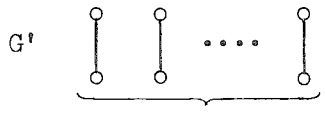
5.13 Theorem: *If* $\underline{\underline{G}}$ *is s-e, then so is* G.

Proof: Suppose the theorem is not true. Then there is a s-e graph H and a graph G of smallest size (for H) that is not s-e but has $\underline{\underline{G}}$ = H.

Let $\pi \in \Xi(G)$. We now proceed to show that π is orbital, thus arriving at a contradiction.

Since G ≠ H, G has at least one point of degree one. We consider two cases:

Case A: Suppose G is of the form



where m ≥ 1 and G' is a (possibly disconnected) graph with no points of degree one.

Then H is clearly the graph

$$G' \quad \underbrace{o \quad o \quad \cdots\cdots \quad o}_{m \text{ copies}} .$$

Since G' has no point of degree one, π can be written in the form

$$\{C_1|C_2|\cdots|C_r|D_1|D_2|\cdots|D_k\} \qquad (0 \leq r \text{ and } 1 \leq k)$$

where

$$\{C_1|C_2|\cdots|C_r\} \in \Xi(G'). \tag{5.6}$$

Consequently,

$$\pi' = \{C_1|C_2|\cdots|C_r|D\} \in \Xi(H)$$

where D is a cell containing the illustrated m isolated points of H. But H is s-e by hypothesis, and so $\pi' \in \Theta(H)$.

Take points v_1 and v_2 such that $v_1 \underset{\pi}{\sim} v_2$. We have two possibilities:

(a) $v_1, v_2 \in C_i$ $(1 \leq i \leq r)$.

Since $\pi' \in \Theta(H)$ there is an element γ' of $\Gamma_{\pi'}(H)$ such that $v_1^{\gamma'} = v_2$.

Defining γ by $v^{\gamma} = \begin{cases} v^{\gamma'} & \text{if } v \in G' \\ v & \text{otherwise} \end{cases}$ we find that $\gamma \in \Gamma_{\pi}(G)$ and $v_1^{\gamma} = v_2$.

(b) $v_1, v_2 \in D_i$ $(1 \leq i \leq k)$.

Since the subgraph F of G induced by D_i is regular (5.7) it consists of copies of $\overset{o}{|}\overset{o}{}$ or of isolated points.

(1) If F consists of copies of $\overset{o}{|}\overset{o}{}$ then D_i is an orbit of $\Gamma_{\pi}(G)$.

(2) If F consists of isolated points, denote by $\bar{v}_1$ and $\bar{v}_2$

the points connected (in G) to v_1 and v_2 respectively. Suppose $\bar{v}_1 \in D_j$. Then $\bar{v}_2 \in D_j$ since

$$d(v_1, D_j) = d(v_2, D_j).$$

Hence the permutation $(v_1v_2)(\bar{v}_1\bar{v}_2)$ is an element of $\Gamma_\pi(G)$ and takes v_1 onto v_2.

Case B: If G is not of case A, then it must have a point $\bar{v}$ of degree one connected to a point of degree greater than one.

Suppose $\bar{v}$ is in the cell C of π. Then all points in C have degree one.

Define $R = \{v \in V | d(v, C) \neq 0\}$.

Let C' be a cell of π s.t. $C' \cap R \neq \phi$. Let $v_1 \in C' \cap R$.

(1) Suppose there is a point v_2 in $C' \backslash R$. Then $d(v_1, C) \neq 0$ but $d(v_2, C) = 0$ which contradicts π being equitable, since $v_1 \underset{\pi}{\sim} v_2$.

(2) Suppose there is a point v_2 in $R \backslash C'$. Then choosing $\bar{v}_1$ and $\bar{v}_2$ in C connected to v_1 and v_2 respectively we find $d(\bar{v}_1, C') = 1$ but $d(\bar{v}_2, C') = 0$ which again gives us a contradiction.

Consequently we must have $R = C'$ so that R is a cell of π.

Suppose R is the set $\{r_1, r_2, \cdots, r_m\}$ $(1 \leq m)$.

Now define $S = \{v \in V | v$ has degree one (in G) and $d(v, R) \neq 0\}$.

Let C' be a cell of π s.t. $C' \cap S \neq \phi$. Let $v_1 \in C' \cap S$. Suppose there is a point v_2 in $C' \backslash S$. Then $d(v_1, R) = 1$ but $d(v_2, R) = 0$ which contradicts π being equitable, since $v_1 \underset{\pi}{\sim} v_2$ and $R \in \pi$. Hence

$C' \subseteq S$, and so S is a union of cells of π.

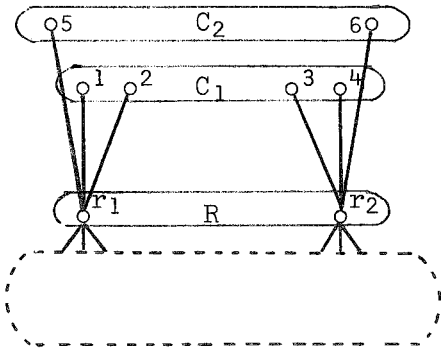
Say $S = C_1 \cup C_2 \cup \cdots \cup C_r$ where each C_i is a cell of π.

For $1 \leq i \leq r$, $1 \leq j \leq m$ define

$$C_i^{\,j} = \{v \in C_i \mid v \text{ is adjacent to } r_j\}.$$

Since $R \in \pi$ and $d(r_j, C_i) = |C_i^{\,j}|$ we find that $|C_i^{\,j}|$ is independent of j.

Our constructions so far may be made clearer by considering the following schematic diagram, where $m = r = 2$.



$S = C_1 \cup C_2$

$C_1^{\,1} = \{1, 2\}$, $C_1^{\,2} = \{3, 4\}$

$C_2^{\,1} = \{5\}$, $C_2^{\,2} = \{6\}$

Figure 5.5

Let G' be the subgraph of G induced by V\S and let $\pi' = \pi\backslash\{C_1, C_2, \cdots, C_r\}$.

Then $\pi' \in \Xi(G')$ by 5.6. Furthermore, $\underline{G'} = \underline{G} = H$ since we have only removed endpoints of G. Hence by our induction hypothesis, $\pi' \in \Theta(G')$.

Now suppose $v_1 \underset{\pi}{\sim} v_2$.

(1) If $v_1, v_2 \notin S$ then $v_1 \underset{\pi'}{\sim} v_2$. Hence there is an element γ' of $\Gamma_{\pi'}(G')$ such that $v_1^{\gamma'} = v_2$. Construct γ as follows:

(a) If $v \in V \backslash S$ let $v^{\gamma} = v^{\gamma'}$.

(b) For $1 \leq i \leq r$, $1 \leq j \leq m$ let γ map C_i^j onto C_i^k where r_j maps onto r_k according to part (a). Any bijective mapping will do.

Then it is easy to see that $\gamma \in \Gamma_{\pi}(G)$, and that γ takes v_1 onto v_2.

(2) If $v_1, v_2 \in C_i$ for some $1 \leq i \leq r$, suppose that $v_1 \in C_i^j$ and $v_2 \in C_i^k$. Then we can find $\gamma' \in \Gamma_{\pi'}(G')$ which takes r_j onto r_k and extend it to $\gamma \in \Gamma_{\pi}(G)$ in the same way as in (1) above. This gives us γ which takes C_i^j onto C_i^k and we can choose γ to take v_1 onto v_2.

So in any case we have $\gamma \in \Gamma_{\pi}(G)$ which takes v_1 onto v_2. Therefore $\pi \in \Theta(G)$. □

5.14 Corollary 1: *All trees and forests are s-e.* □

5.15 Corollary 2: *For a graph* G *define* a(G) *to be the complement of* $\underline{G}$ *with any isolated points (of* $\underline{G}$*) removed. Then if* $a^r(G)$ *is null or s-e for any* r*, then* G *is s-e.*

Proof: By applying 5.13 to G and $\bar{G}$ and using 5.8. □

Let $\pi \in \Pi(V)$ and $C_1, C_2 \in \pi$. Then we say that C_1 is *trivially joined* to C_2 (in G) if one of the following holds for all $v \in C_1$.

(1) $d(v, C_2) = 0$.

(2) $C_1 \neq C_2$ and $d(v, C_2) = |C_2|$.

(3) $C_1 = C_2$ and $d(v, C_2) = |C_2| - 1$.

Obviously, if $\pi \in \Xi(G)$, then a trivial cell of π is trivially joined to every cell of π.

The following result is well known, for example to Levi [40].

5.16 Lemma: *Let* $\pi \in \Pi(V)$ *and let* $C \in \pi$ *be trivially joined to every cell of* $\pi\backslash C$. *If* G_1 *and* G_2 *are respectively the subgraphs of* G *induced by* C *and* $V\backslash C$ *then* $\Gamma_\pi(G) = \Gamma(G_1) \oplus \Gamma_{\pi\backslash C}(G_2)$.

Proof: Obviously $\Gamma_\pi(G) \leq \Gamma(G_1) \oplus \Gamma_{\pi\backslash C}(G_2)$. Suppose $\gamma_1 \in \Gamma(G_1)$ and $\gamma_2 \in \Gamma_{\pi\backslash C}(G_2)$, and define $\gamma = \gamma_1\gamma_2$. Suppose $\{v_1, v_2\}$ is an edge of G. If v_1 and v_2 are both in C or both not in C, then obviously $\{v_1^\gamma, v_2^\gamma\}$ is an edge of G. If $v_1 \in C$ but $v_2 \notin C$, then $\{v_1^\gamma, v_2^\gamma\}$ is an edge of G because C is trivially joined to the cell of π containing v_2 and v_2^γ. Thus $\gamma \in \Gamma_\pi(G)$. □

5.17 Theorem: *Let* $\pi \in \Xi(G)$, $C \in \pi$ *and suppose that*

(1) *The subgraph of* G *induced by* C *is transitive.*

(2) *For any* $C' \in \pi\backslash C$, *then either* $(|C|, |C'|) = 1$ *or* $|C| = |C'| = 2$.

Then C *is an orbit of* Γ_π.

Proof:

(1) Suppose $|C| > 2$. Let $C' \in \pi\backslash C$.

Then counting the edges joining C to C', we have

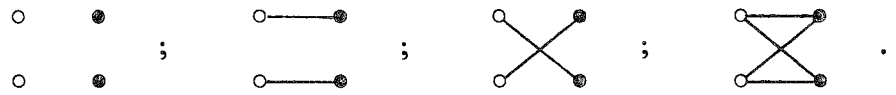
$$|C|d(v, C') = |C'|d(v', C) \text{ for any } v \in C,\ v' \in C'.$$

But $|C|$ and $|C'|$ are coprime and so $|C|$ divides $d(v', C)$ and $|C'|$ divides $d(v, C')$. However, $0 \leq d(v', C) \leq |C|$ and

$0 \leq d(v, C') \leq |C'|$. Hence either $d(v, C') = d(v', C) = 0$ or $d(v, C') = |C'|$ and $d(v', C) = |C|$. Consequently, C satisfies the requirements of 5.16 and so C is an orbit of Γ_π since $\langle C \rangle$ is transitive.

(2) Suppose the cells of size 2 of π are $C_1, C_2, \cdots, C_k$ where for $1 \leq i \leq k$, $C_i = \{v_i{}^1 v_i{}^2\}$.

Two cells of size 2 can be equitably joined in one of these four ways:



Since $\langle C_i \rangle$ is either K_1 or $\bar{K}_2$ for $1 \leq i \leq k$ we find that the permutation

$$\gamma = (v_1{}^1 v_2{}^1)(v_1{}^2 v_2{}^2)\cdots(v_1{}^k v_2{}^k) \text{ is in } \Gamma_\pi.$$

Hence each C_i is an orbit of Γ_π.

(3) If $|C| = 1$ then the theorem is trivial. □

5.18 Corollary: *If the conditions of the theorem are satisfied for each cell C of π then $\pi \in \Theta(G)$.* □

5.19 Theorem: *Let $\pi \in \Xi(G)$ have ℓ cells, where $n - \ell \leq 5$. Let C be a cell of π of the smallest non-trivial size. Then C is an orbit of Γ_π. The bound is sharp.*

Proof: The possible sizes for the non-trivial cells of π are

2,

22, 3,

222, 23, 4,

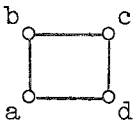
2222, 223, 33, 24, 5,

22222, 2223, 233, 224, 34, 25, 6.

Theorem 5.17 can be applied to all these cases except 24 and 224 since all the regular graphs on ≤ 6 points are transitive. Hence we need consider only the cases where π has a cell C_1 consisting of 4 points and one or two cells consisting of two points each.

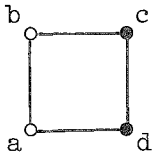
For a start we notice that if π has a cell of size 2 joined trivially to C_1 then that cell is an orbit of Γ_π by the same argument as used in 5.17. In this situation the case 24 is proven and the case 224 reduces to the case 24.

Consider the subgraph $\langle C_1 \rangle$. Since it is regular (5.7), it must be K_4, Z_4 or the complement of one of these. Hence it can be labelled so that its automorphism group contains the group D_4 of the square

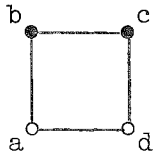


Suppose $C_2 = \{v_1, v_2\}$ is a cell of π which is not trivially joined to C_1. Then since C_2 is equitably joined to C_1 we must have $d(v_1, C_1) = d(v_2, C_1) = 2$, and $d(v, C_2) = 1$ for any $v \in C_1$. Hence we can bisect C_1 into two halves -- those points adjacent to v_1 and those adjacent to v_2. This can be done in one of three non-equivalent ways:

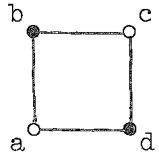
A:



B:



C:



Define subsets D_A, D_B, D_C of D_4:

$D_A = \{(bc)(ad), (bd)(ac)\}$

$D_B = \{(ab)(cd), (ac)(bd)\}$

$D_C = \{(abcd), (adcb), (ab)(cd), (ad)(bc)\}$.

Note that D_A consists of those elements of D_4 which swap the two halves of C_1 shown in A, and similarly for D_B and D_C.

(i) In the case 24, Γ_π contains $(v_1v_2)\gamma$ where γ is from the set D_A, D_B or D_C depending on the way C_2 is joined to C_1.

(ii) Let $C_3 = \{v_3, v_4\}$ be another cell of π, not joined trivially to C_1. Now suppose that for example C_2 gives the bisection A of C_1 and C_3 gives the bisection B. Then we let $\gamma \in D_A \cap D_B$. This can always be done since any pair of the sets D_A, D_B and D_C have an element in common.

Now C_2 and C_3 can only be joined in one of the ways shown in 5.17, and so the permutation $(v_1v_2)(v_3v_4)\gamma$ is in Γ_π, which shows that both C_2 and C_3 are orbits of Γ_π.

The graph marked (*) in Figure 5.2 is a counter-example where $n - \ell = 6$. □

5.20 Theorem: *Let* $\pi \in \Xi(G)$ *where* G *is connected and let* $C \in \pi$. *Let* π_C *be the partition of the points of* G *into cells of equal distance from* C. *Then* $\pi \leq \pi_C$.

Proof: For any $j \geq 0$, let $D_j = \{v \in V | \partial(v, C) = j\}$.

We prove by induction on j that each D_j is a union of cells of π.

Firstly, $D_0 = C \in \pi$.

Now suppose that for some $j \geq 0$, D_j is a union of cells of π.

Then whenever $v_1 \underset{\pi}{\sim} v_2$ we have $d(v_1, D_j) = d(v_2, D_j)$.

Now $D_{j+1} = \{v \in V | \partial(v, C) > j \text{ and } d(v, D_j) \neq 0\}$.

Therefore, if $v_1 \in D_{j+1}$, we must have $v_2 \in D_{j+1}$ and so D_{j+1} is also a union of cells of π.

Hence $\pi \leq \pi_C$. □

If G is transitive and $\pi_C \in \Xi(G)$ whenever $|C| = 1$, then G is called *distance-regular*. Distance regularity is also defined for non-transitive graphs. If $\pi_C \in \Theta(\Gamma)$ in the same circumstances, G is called *distance-transitive*. See Biggs [6] for further details.

5.21 Let $\pi \in \Pi(V)$ be the partition $\{C_1 | C_2 | \cdots | C_\ell\}$ and let the elements of V be $v_1, v_2, \cdots, v_n$ in some order. We define an $\ell \times n$ matrix $T = T(\pi)$ by

$$T_{ij} = \begin{cases} 1 & \text{if } v_j \in C_i \\ 0 & \text{otherwise.} \end{cases}$$

Clearly TT' is the $\ell \times \ell$ diagonal matrix $\mathrm{diag}(|C_1|, \cdots, |C_\ell|)$ and so $(TT')^{-1}$ exists. Let A be the adjacency matrix of G with the labelling $v_1, \cdots, v_n$, and define the $\ell \times \ell$ matrix

$$B = TAT'(TT')^{-1}.$$

5.22 Theorem: $\pi \in \Xi(G)$ *iff* $TA = BT$. *Furthermore, if* $\pi \in \Xi(G)$, *then* $B_{ij} = d(v, C_i)$ *for any* $v \in C_j$, *where* $1 \leq i \leq \ell$, $1 \leq j \leq \ell$.

Proof: Suppose $TA = BT$. Then for $1 \leq i \leq \ell$, $1 \leq j \leq n$ we have

$$(TA)_{ij} = \sum_{k=1}^{n} T_{ik}A_{kj}$$

$$= \sum_{k \in C_i} A_{kj}$$

$$= d(v_j, C_i)$$

and

$$(BT)_{ij} = \sum_{k=1}^{\ell} B_{ik}T_{kj}$$

$$= B_{ik}T_{kj} \text{ where } v_j \in C_k$$

$$= B_{ik}.$$

Hence for $v_{j_1}, v_{j_2} \in C_k$ $(1 \leq k \leq \ell)$, we have $d(v_{j_1}, C_i) = B_{ik} = d(v_{j_2}, C_i)$ $(1 \leq i \leq \ell)$ and so $\pi \in \Xi(G)$.

Conversely, if $\pi \in \Xi(G)$ define the $\ell \times \ell$ matrix $\tilde{B}$ by

$$\tilde{B}_{ij} = d(v, C_i) \text{ where } v \in C_j \ (1 \leq i \leq \ell, 1 \leq j \leq \ell).$$

Then, in a similar fashion, $TA = \tilde{B}T$, and this implies that

$\tilde{B} = TAT'(TT')^{-1} = B.$ □

5.23 If $\pi \in \Xi(G)$, then the matrix B is sometimes called the *quotient matrix* of A induced by π, although some authors use this title for the transpose B'. It plays an important part in many algorithms for graph isomorphism, for example that of Corneil and Gotlieb [11, 14].

The matrix B also plays a central role in many other regions of graph theory, in particular spectral theory. For example, the characteristic polynomial of B divides that of A, a result first proved by Haynsworth [23]. We shall not be concerned with these matters here. For further information see Sachs, Petersdorff and Finck [19, 53, 60, 61], Schwenk [63] and Djokovic [15].

5.24 We turn now to the problem of computing $\xi(\pi)$. This problem is of central importance in many proposed algorithms for graph isomorphism for the following reason. Given any partition $\pi \in \Pi(V)$ we have (by 5.10)

$$\theta(\pi) \leq \xi(\pi) \leq \pi.$$

Consequently $\xi(\pi)$ is in general a better estimate of $\theta(\pi)$ than is π, and in many cases (s-e graphs for example) will equal $\theta(\pi)$ exactly. $\xi(\pi)$ can often be used in place of $\theta(\pi)$, which is much harder to compute.

5.25 In order to store a partition in the computer we need to assign an order to the cells. Similarly, we need to label the points of a graph. These matters have been discussed in sections 3.8 to 3.10. They lead us to the following definitions.

G is a labelled graph with points $V = \{1, 2, \cdots, n\}$.

$\tilde{\Pi}(V)$ is the class of *ordered partitions* of V. In other words, $\tilde{\Pi}(V)$ is the class of sequences $[C_1|C_2|\cdots|C_k]$ where $\{C_1|C_2|\cdots|C_k\} \in \Pi(V)$.

If $\pi \in \tilde{\Pi}(V)$, then $\pi(i)$ denotes the i-th cell of π.

If π_1 and π_2 are in either $\Pi(V)$ or $\tilde{\Pi}(V)$ we write $\pi_1 \simeq \pi_2$ to indicate that π_1 and π_2 have the same cells in some order.

Otherwise when relations and functions defined on $\Pi(V)$ are applied to elements of $\tilde{\Pi}(V)$ we understand that the corresponding unordered partitions are intended. For example, if $\tilde{\pi}_1, \tilde{\pi}_2 \in \tilde{\Pi}(V)$, then by $\tilde{\pi}_1 \leq \tilde{\pi}_2$ we mean that $\pi_1 \leq \pi_2$ where $\pi_1, \pi_2 \in \Pi(V)$, $\pi_1 \simeq \tilde{\pi}_1$ and $\pi_2 \simeq \tilde{\pi}_2$.

5.26 We begin by presenting a very simple algorithm for computing $\xi(\pi)$. The ideas behind this algorithm date back to Duijvestijn [17], Unger [76] and Morgan [46]. More recently, it has been proposed in a similar form by Corneil [11], Parris [50], Steen [69] and Tinhofer [72].

Suppose $\pi = [C_1|\cdots|C_k] \in \tilde{\Pi}(V)$. Then we define the vector $\underset{\sim}{d}(v, \pi)$ for each $v \in V$ by

$$\underset{\sim}{d}(v, \pi) = [d_0, d_1, \cdots, d_k]$$

where $$v \in \pi(d_0)$$

and $$d_i = d(v, C_i) \quad (1 \leq i \leq k).$$

Note that all degrees are taken in the graph G.

5.27 Algorithm: $\mathcal{R}_*$. Compute $\tilde{\pi} \simeq \xi(\pi)$.

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

(2) Compute $\underset{\sim}{d}(v, \tilde{\pi})$ for each $v \in V$.

(3) Set π_1 to the ordered partition of V whose cells contain points for which $\underset{\sim}{d}(v, \tilde{\pi})$ is equal, and are ordered according to a lexicographic ordering of these vectors.

(4) If $\pi_1 \simeq \tilde{\pi}$ stop.

(5) Set $\tilde{\pi} := \pi_1$. Go to step (2).

Let $\mathcal{R}_*(\pi)$ denote the value of $\tilde{\pi}$ when the algorithm stops.

5.28 Example: Let G be the graph of Figure 5.6 and let $\pi = [1, 2, 3, 4, 5, 6, 7, 8]$.

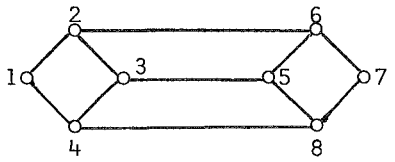


Figure 5.6

(1) $\tilde{\pi} = [1, 2, 3, 4, 5, 6, 7, 8]$.

(2) $\underset{\sim}{d}(v, \tilde{\pi}) = [1, 2], [1, 3], [1, 3], [1, 3],$
$[1, 3], [1, 3], [1, 2], [1, 3],$
for $v = 1, 2, \cdots, 8$ respectively.

(3) $\pi_1 = [1, 7|2, 3, 4, 5, 6, 8]$.

(5) $\tilde{\pi} = [1, 7|2, 3, 4, 5, 6, 8]$.

(2) $\underset{\sim}{d}(v, \tilde{\pi}) = [1, 0, 2], [2, 1, 2], [2, 0, 3], [2, 1, 2],$

$[2, 0, 3], [2, 1, 2], [1, 0, 2], [2, 1, 2],$

for $v = 1, 2, \cdots, 8$ respectively.

(3) $\pi_1 = [1, 7|3, 5|2, 4, 6, 8]$.

(5) $\tilde{\pi} = [1, 7|3, 5|2, 4, 6, 8]$.

(2) $\underset{\sim}{d}(v, \tilde{\pi}) = [1, 0, 0, 2], [3, 1, 1, 1], [2, 0, 1, 2], [3, 1, 1, 1],$

$[2, 0, 1, 2], [3, 1, 1, 1], [1, 0, 0, 2], [3, 1, 1, 1],$

for $v = 1, 2, \cdots, 8$ respectively.

(3) $\pi_1 = [1, 7|3, 5|2, 4, 6, 8]$.

(4) Stop.

Hence $\mathcal{R}_*(\pi) = [1, 7|3, 5|2, 4, 6, 8]$.

5.29 Theorem: $\mathcal{R}_*(\pi) \simeq \xi(\pi)$ *for any* $\pi \in \tilde{\Pi}(V)$.

Proof: Consider step (3). The first element of $\underset{\sim}{d}(v, \tilde{\pi})$ for each v ensures that $\pi_1 \leq \tilde{\pi}$. Also, the condition $\pi_1 \simeq \tilde{\pi}$ is just that for $\tilde{\pi}$ to be equitable.

Hence $\mathcal{R}_*(\pi)$ is equitable, and $\mathcal{R}_*(\pi) \leq \pi$. Therefore, $\mathcal{R}_*(\pi) \leq \xi(\pi)$.

Let C be a cell of $\xi(\pi)$ and let $v_1, v_2 \in C$. Now $\xi(\pi) \leq \pi$ and so when step (2) is executed for the first time each cell of $\tilde{\pi}$ is a union of cells of $\xi(\pi)$ (by 2.10). Therefore $\underset{\sim}{d}(v_1, \tilde{\pi}) = \underset{\sim}{d}(v_2, \tilde{\pi})$

and so v_1 and v_2 will be in the same cell of π_1 after executing step (3). Hence, at this stage $\xi(\pi) \leq \pi_1$. Repeating this argument for each time steps (2) and (3) are executed we see that $\xi(\pi) \leq \mathcal{R}_*(\pi)$, which completes the proof. □

5.30 Despite its simplicity the algorithm $\mathcal{R}_*$ has several disadvantages:

(1) We are required to sort vectors of varying lengths. In King's implementation [31] this problem is simplified by a process of "compacting" the vectors. For example, if $\underset{\sim}{d}(v, \tilde{\pi}) = [12, 01, 03, 07]$ then we can write this as an integer 12010307. However, special handling is still required as such integers can be much too large to store as integers in the normal way.

(2) Much unnecessary computation is performed. For example, suppose that after step (3) the partitions $\tilde{\pi}$ and π_1 have a common cell C. Then for all v_1, v_2 in the same cell of π_1 we have $d(v_1, C) = d(v_2, C)$. Therefore there is no need to compute these degrees next time step (2) is executed since they will make no difference to the sorting.

5.31 A few improvements to $\mathcal{R}_*$ have been suggested in special cases. If the initial partition π contains a trivial cell, say $\{v\}$, then Saucier [62] first divides V into cells of equal distance from v (compare 5.20). Then we know that cells C_1 and C_2 are trivially joined if $|\partial(v, C_1) - \partial(v, C_2)| > 1$. This would seem to save much time if the graph has a large diameter. Another variation is used by Levi [40] for the fundamentally different case where the cells of the partitions contain both the points and the edges of the graph.

We now present a new algorithm which, while not the final answer to the problem, has been found to work very satisfactorily in practice.

Let $\pi \in \tilde{\Pi}(V)$ have r cells. Let K be a positive integer. The value of K will be discussed later.

5.32 Algorithm: $\mathcal{R}_K$: Compute $\tilde{\pi} \simeq \xi(\pi)$.

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

$k := K$;

$\ell := r$;

$\ell' := r$.

(2) If $k > \ell$ or $\ell = n$, *stop*.

(3) Set $C := \tilde{\pi}(k)$;

$i := 1$.

(4) If $|\tilde{\pi}(i)| = 1$ go to step (8).

(5) Sort $\tilde{\pi}(i)$ into cells $C_1, C_2, \cdots, C_s$ according to $d(v, C)$ for $v \in \tilde{\pi}(i)$.

(6) If $s = 1$, go to step (8).

(7) Set $\tilde{\pi}(i) := C_1$. For $2 \leq j \leq s$ set $\tilde{\pi}(\ell' + j - 1) := C_j$. Set $\ell' := \ell' + s - 1$.

(8) Set $i := i + 1$. If $i \leq \ell$, go to step (4).

(9) Set $\ell := \ell'$;

$k := k + 1$. Go to step (2).

Let $\mathcal{R}_K(\pi)$ denote the value of $\tilde{\pi}$ (with ℓ cells) when the algorithm stops.

5.33 We consider the example of 5.28 and apply algorithm 5.32 with K = 1.

(1) $\tilde{\pi} = [1, 2, 3, 4, 5, 6, 7, 8]$, $k = \ell = \ell' = 1$.

(3) $C = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $i = 1$.

(5) $d(v, C) = 2, 3, 3, 3, 3, 3, 2, 3$ for $v = 1, 2, \cdots, 8$ respectively. $C_1 = \{1, 7\}$, $C_2 = \{2, 3, 4, 5, 6, 8\}$.

(7) $\tilde{\pi} = [1, 7|2, 3, 4, 5, 6, 8]$, $\ell' = 2$.

(9) $\ell = 2$, $k = 2$.

(3) $C = \{2, 3, 4, 5, 6, 8\}$, $i = 1$.

(5) $d(1, C) = 2$, $d(7, C) = 2$ so $s = 1$.

(8) $i = 2$.

(5) $d(v, C) = 2, 3, 2, 3, 2, 2$ for $v = 2, 3, 4, 5, 6, 8$ respectively. $C_1 = \{2, 4, 6, 8\}$, $C_2 = \{3, 5\}$.

(7) $\tilde{\pi} = [1, 7|2, 4, 6, 8|3, 5]$, $\ell' = 3$.

(9) $\ell = 3$, $k = 3$.

(3) $C = \{3, 5\}$, $i = 1$.

(5) $d(1, C) = d(7, C) = 0$ so $s = 1$.

(8) $i = 2$.

(3) $d(2, C) = d(4, C) = d(6, C) = d(8, C) = 1$ so $s = 1$.

(8) $i = 3$.

(5) $d(3, C) = d(5, C) = 1$ so $s = 1$.

(9) $\ell = 3$, $k = 4$.

(2) $k > \ell$ so stop: $\mathcal{R}_1(\pi) = [1, 7|2, 4, 6, 8|3, 5]$.

5.34 Theorem: *For any* $\pi \in \tilde{\Pi}(V)$, $\mathcal{R}_1(\pi) \simeq \xi(\pi)$.

Proof:

(1) For each value of k, steps (4) to (8) are executed less than n times. Furthermore, k is incremented at step (9) and stops execution when it passes ℓ. Hence the algorithm terminates and so $\mathcal{R}_1(\pi)$ is defined.

(2) The partition $\tilde{\pi}$ is altered only at step (7) where it is made finer. Let C be a cell of $\xi(\pi)$ and let $v_1, v_2 \in C$. At step (1) we set $\tilde{\pi}$ to π, which is coarser than $\xi(\pi)$.

Suppose that $\tilde{\pi}$ is coarser than $\xi(\pi)$ just before step (7) is executed. By 2.10 each cell of $\tilde{\pi}$ is a union of cells of $\xi(\pi)$. Therefore $d(v_1, C) = d(v_2, C)$ and so v_1 and v_2 will be in the same cell of $\tilde{\pi}$ after step (7) is executed.

Hence $\xi(\pi) \leq \mathcal{R}_1(\pi) \leq \pi$.

(3) Suppose $\mathcal{R}_1(\pi)$ is not an equitable partition.

Then $\mathcal{R}_1(\pi)$ contains cells C_1 and C_2 and points $v_1, v_2 \in C_1$ such that

$$d(v_1, C_2) \neq d(v_2, C_2).$$

Since the partition $\tilde{\pi}$ is made successively finer during the execution of the algorithm, v_1 and v_2 must always be in the same cell of $\tilde{\pi}$. We show that this leads to a contradiction.

(a) Suppose that before and after some execution of step (7), C_2 is contained in $\tilde{\pi}(p)$ and $\tilde{\pi}(q)$ respectively. Then clearly $q \geq p$, and also $q \leq n$. However, k is set to 1 initially and is incremented by 1 at each execution of step (9).

Therefore at some execution of step (3) we have $C_2 \subseteq \tilde{\pi}(k)$.

(b) Since we are assuming v_1 and v_2 are not separated, we must have

$$d(v_1, \tilde{\pi}(k)) = d(v_2, \tilde{\pi}(k)).$$

But $d(v_1, C_2) \neq d(v_2, C_2)$ and $C_2 \subseteq \tilde{\pi}(k)$. Therefore, there is at least one cell, say C_3, of $\mathcal{R}_1(\pi)$ which is contained in $\tilde{\pi}(k) \backslash C_2$ and such that $d(v_1, C_3) \neq d(v_2, C_3)$.

(c) Since C_2 and C_3 are distinct cells of $\mathcal{R}_1(\pi)$ they must be separated at some execution of step (7). At least one of them, say C_2, will then be a subset of some cell $\tilde{\pi}(j)$ where $j > k$.

(d) As in (a), some cell containing C_2 will again be encountered as $\tilde{\pi}(k)$ at step (3).

Clearly the argument from (a) to (d) can be repeated indefinitely and so the algorithm will never terminate. This contradicts (1).

Therefore, $\mathcal{R}_1(\pi)$ is equitable, and so $\mathcal{R}_1(\pi) \simeq \xi(\pi)$ by part (2). □

5.35 One of the greatest advantages that algorithm 5.32 has over algorithm 5.27 is that in many cases of practical concern the constant K can be set to a value greater than one, without destroying the validity of the algorithm. We now give a method for setting K which will later be seen to have an important application.

Let $\pi_1 \in \tilde{\Pi}(V)$ be an equitable partition coarser than π. Suppose π_1 has ℓ_1 cells. Let q be an integer $(1 \leq q \leq \ell_1)$ such that for $1 \leq i \leq q$, $\pi(j) \subseteq \pi_1(i)$ for at most one $\pi(j)$, $(1 \leq j \leq q)$.

5.36 Theorem: $\mathcal{R}_K(\pi) \simeq \xi(\pi)$ *if* $K = q + 1$.

Proof:

(1) By the same arguments as for 5.34 the algorithm terminates and

$$\xi(\pi) \leq \mathcal{R}_K(\pi) \leq \pi.$$

(2) Suppose $\mathcal{R}_K(\pi)$ is not equitable.

Then $\mathcal{R}_K(\pi)$ contains a cell C_1 such that for some two points v_1, v_2 in the same cell of $\mathcal{R}_K(\pi)$, we have $d(v_1, C_1) \neq d(v_2, C_1)$.

Let π_1 be the equitable partition defined above. Since $\mathcal{R}_K(\pi) \leq \pi_1$, there is a cell C of π_1 of the form $C_1 \cup C_2 \cup \cdots \cup C_s$ where each C_i is a cell of $\mathcal{R}_K(\pi)$.

But π_1 is equitable and so $d(v_1, C) = d(v_2, C)$. Therefore at least one of the cells C_i $(2 \leq i \leq s)$ also has $d(v_1, C_i) \neq d(v_2, C_i)$. Say $i = 2$.

Hence the defined relationship between π and π_1 ensures that, if C_1 and C_2 are contained in different cells of π, one of them,

say C_2, is contained in a cell $\pi(j)$ where $j \geq K$. In this case we can take up the proof of 5.34 at step (a) and derive the same contradiction.

If however C_1 and C_2 are contained in the same cell of π we can take up the proof of 5.34 at step (c), where we read C_1 for C_3.

In either case we conclude that $\mathcal{R}_K(\pi)$ is equitable, and so $\mathcal{R}_K(\pi) = \xi(\pi)$. □

5.37 We now study the efficiency of algorithms 5.27 and 5.32, for the data structures described in sections 3.8-3.10. In both algorithms the time taken for indexing etc., is quite trivial and so we may accurately write

$$t_1 = N_1 d_1 + s_1 \quad \text{for} \quad \mathcal{R}_*$$

or

$$t_2 = N_2 d_2 + s_2 \quad \text{for} \quad \mathcal{R}_K$$

where t_i is the total time, N_i is the number of times we must compute $d(v, C)$ for some point v and cell C, d_i is the average time for such a computation, and s_i is the time taken in sorting, for $i = 1, 2$.

Suppose π and $\xi(\pi)$ have ℓ_0 and ℓ_1 cells respectively.

5.38 Consider algorithm 5.27. Let p be the number of times step (2) is executed. Since $p \leq n$ and $p = \left[\frac{n+1}{2}\right]$ when $G = P_n$ we see that $p = O(n)$ in the worst cases.

(1) At the j-th execution of step (2), $\tilde{\pi}$ has a least $\ell_0 + j - 1$ cells.

$$\text{Therefore } N_1 \geq \sum_{j=1}^{p} n(\ell_0 + j - 1) = \tfrac{1}{2}np(2\ell_0 + p - 1).$$

(2) At the j-th execution of step (2) we are required to sort n vectors of length at least $\ell_0 + j$. Even if a very efficient means of packing the vectors is used the time for sorting will be at least of order n log n, [20].

Therefore $s_1 = O(pn \log n)$

$= O(n^2 \log n)$ (at least) in the worst cases.

5.39 Consider algorithm 5.32 for some value of K. Clearly step (3) is executed $\ell_1 - K + 1$ times.

(1) For each value of k we must compute $d(v, \tilde{\pi}(k))$ for at most n points, depending on step (4).

Therefore $N_2 \leq n(\ell_1 - K + 1)$.

(2) Sorting is performed at step (5) where we must order the points of $\tilde{\pi}(i)$ according to their degree relative to C. Now $0 \leq d(v, C) \leq n - 1$ for any $v \in V$, $C \subseteq V$. This enables us to use the address-calculation sort (see [20]). This sorting method is not only the fastest but the simplest. The time it takes is of order $|\tilde{\pi}(i)|$ and so the time taken in sorting for each value of k is of order

$$\sum_{i=1}^{\ell} |\pi(i)| = n.$$

Therefore $s_2 = O(n(\ell_1 - K + 1))$.

5.40 In the author's implementation a computation of the form $d(v, C)$ takes a fixed time since the population count instruction can be used (see 3.2). Therefore we can say that in the worst cases we have

$t_1 = O(n^3)$ while $t_2 = O(n(\ell_1 - K + 1))$

$= O(n^2)$ for fixed K.

The expression for t_2 again emphasizes the advantage in being able to set $K > 1$ in some cases.

5.41 The efficiency of algorithm 5.32 has been examined extensively for "random" graphs of the type described in Section 3.11. The results for the case where the initial partition is the unit partition are illustrated in Figure 5.7. Each point represents the average time for about 100 graphs. The cases where $\sigma = 0{\cdot}75$ or $\sigma = 0{\cdot}50$ are seen to be very nearly linear.

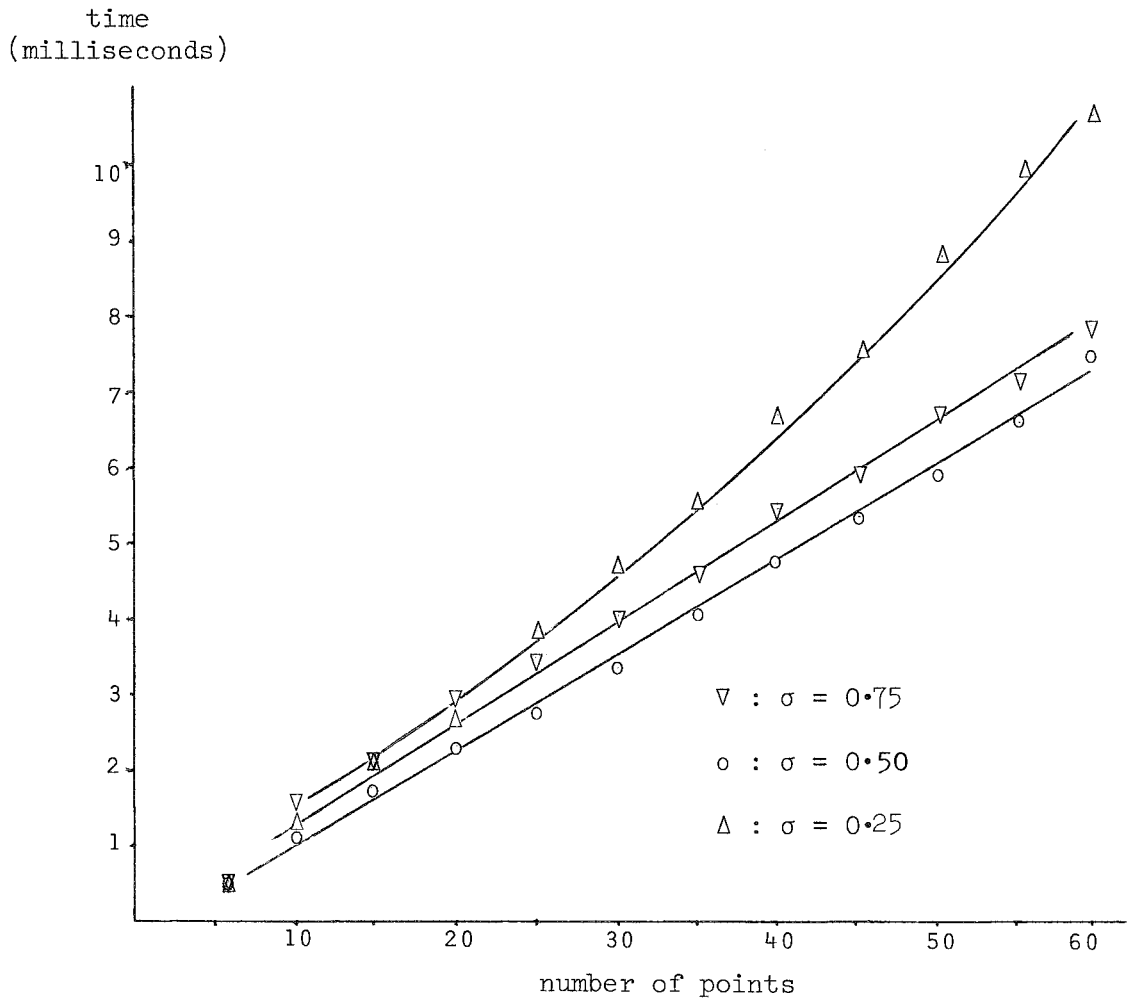


Figure 5.7

CHAPTER SIX

BACKTRACK PROGRAMMING - I

6.1 A large proportion of computing tasks in combinatorics can only be handled by something which amounts to an exhaustive search through a large set of possibilities. The most widely employed method for performing such a search in a systematic fashion is known as "backtrack programming" or "depth-first searching". Descriptions of backtrack programming with various degrees of generality can be found in Golomb and Baumert [22], Wells [78], Tarjan [71] or Fillmore and Williamson [18].

We begin this chapter by giving a formal description of backtrack programming as applied to a problem of finding sequences satisfying a given property. This gives us a program with a natural tree-like structure which we then explore.

Following these basic results, which are well known, we introduce the invariance group T of the program and prove some of its properties. It is seen that the automorphism group of the graph, group or whatever object is under consideration is a subgroup of T under certain very common conditions. The invariance group does not appear to have been defined before, although some properties of certain of its subgroups have been utilised. We show that knowledge of a subgroup of T enables us to considerably reduce the amount of work required by the backtrack method.

6.2 Let V be the set $\{1, 2, \cdots, n\}$. Then for $0 \leq k \leq n$ define $Q^{(k)}(V)$ to be the set of sequences $[v_1, \cdots, v_k]$ of distinct elements of V. If $k = 0$ then the symbol $[v_1, \cdots, v_k]$ indicates the null sequence $[\,]$.

Define $Q(V) = \bigcup_{k=0}^{n} Q^{(k)}(V)$. Let P be a property defined on $Q^{(n)}(V)$ and let $U = \{\tau \in Q^{(n)}(V) | \tau \text{ has property P}\}$. We shall direct our attention to the problem of finding U when P is given. For example, if G is a graph with points V, then we might say that $\tau = [v_1, \cdots, v_n]$ satisfies P if $v_1, v_2, \cdots, v_n, v_1$ is a Hamiltonian cycle of G.

One possible way to determine U is by testing each of the n! elements of $Q^{(n)}(V)$ to see which of them satisfy P. However, this technique is obviously impractical except for very small values of n, and so some more efficient means is required. The success of the "backtrack" process lies in its capability for eliminating elements of $Q^{(n)}(V)$ without examining them explicitly. To continue our example, if v_1, v_2, v_3 is a path in G, but v_4 is not connected to v_3, then U contains no elements of the form $[v_1, v_2, v_3, v_4, \cdots, v_n]$.

6.3 If $\nu = [v_1, \cdots, v_k] \in Q(V)$, define $X_\nu = \{v \in V | U \text{ contains an element of the form } [v_1, \cdots, v_k, v, \cdots]\}$. Let $W : Q(V) \to 2^V$ be any function so that for $\nu = [v_1, \cdots, v_k] \in Q(V)$ we have

6.4 $$X_\nu \subseteq W(\nu) \subseteq V \backslash \{v_1, \cdots, v_k\}.$$

The backtrack algorithm we now present produces all sequences $[v_1, \cdots, v_n]$ such that for $1 \leq i \leq n$, $v_i \in W([v_1, \cdots, v_{i-1}])$. The condition 6.4 shows that every element of U is of the form. In practice a trade-off will usually be necessary between the size of $W(\nu)$ and the effort expended in computing it. If $W(\nu) = X_\nu$ for all ν, then only elements of U will be produced. At the other extreme, if $W([v_1, \cdots, v_k]) = V \backslash \{v_1, \cdots, v_k\}$ for all $[v_1, \cdots, v_k]$, then the whole of $Q^{(n)}(V)$ will be produced.

6.5 Algorithm: Find U given P

(1) Set k := 0.

(2) Set U_k := W([v_1, •••, v_k]).

(3) If $U_k = \phi$, go to step (7).

(4) Choose and delete any element v_{k+1} from U_k. Set k := k + 1.

(5) If k < n, go to step (2).

(6) Output [v_1, •••, v_n] if it satisfies P.

(7) Set k := k - 1. If k ≥ 0, go to step (3); otherwise stop.

6.6 We illustrate this algorithm by continuing our example of finding Hamiltonian cycles in a graph G, namely the graph of Figure 6.1.

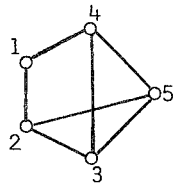


Figure 6.1

Define W as follows:

(1) W([ ]) = V (†)

(2) W([v_1, •••, v_k]) = {v ∈ V\{v_1, •••, v_k} | v is connected to v_k in G} (k ≥ 1).

(†) We have set W([ ]) = V because it makes our example more instructive in later sections. In practice, we would set W([ ]) = {1} to avoid each cycle appearing 5 times. For a more sophisticated algorithm for finding cycles in a graph see Johnson [29].

6.7 We follow the progress of the algorithm until it finds the first solution.

(1) $k = 0$

(2) $U_0 = \{1, 2, 3, 4, 5\}$

(3) $U_0 \neq \phi$

(4) $v_1 = 1$, $U_0 = \{2, 3, 4, 5\}$, $k = 1$

(5) $k < n$ so go to (2)

(2) $U_1 = \{2, 4\}$

(3) $U_1 \neq \phi$

(4) $v_2 = 2$, $U_1 = \{4\}$, $k = 2$

(5) $k < n$ so go to (2)

(2) $U_2 = \{3, 5\}$

(3) $U_2 \neq \phi$

(4) $v_3 = 3$, $U_2 = \{5\}$, $k = 3$

(5) $k < n$ so go to (2)

(2) $U_3 = \{4, 5\}$

(3) $U_3 \neq \phi$

(4) $v_4 = 4$, $U_3 = \{5\}$, $k = 4$

(5) $k < n$ so go to (2)

(2) $U_4 = \{5\}$

(3) $U_4 \neq \phi$

(4) $v_5 = 5$, $U_4 = \phi$, $k = 5$

(5) k = n

(6) [1, 2, 3, 4, 5] *is not* a Hamiltonian cycle

(7) k = 4; go to (3)

(3) $U_4 = \phi$ so go to (7)

(7) k = 3; go to (3)

(4) $v_4 = 5$, $U_3 = \phi$, k = 4

(5) k < n so go to (2)

(2) $U_4 = \{4\}$

(3) $U_4 \neq \phi$

(4) $v_5 = 4$, $U_4 = \phi$, k = 5

(5) k = n

(6) [1, 2, 3, 5, 4] *is* a Hamiltonian cycle

etc.

6.8 This process can be conveniently described in terms of a *program tree* T as shown (for our example) in Figure 6.2. The points of the tree are called *nodes*. The node at the top of the tree is called its *root* and corresponds to the start of the algorithm. The other nodes correspond to a choice of v_{k+1} at step (4) of the algorithm. Each node is considered to be labelled with the sequence $[v_1, \cdots, v_k]$ which is current *after* step (4) has been completed. For clarity, however, only the value of v_k is shown in Figure 6.2. Thus the label of the node marked A is [3, 2, 1, 4]. The algorithm 6.5 begins at the root of the tree and works downwards where possible, taking the left-most branches on the way down (hence the phrase "depth-first"). If it reaches a dead-end, it "backtracks" to find another path downwards, and thus continues until it has traversed the entire tree.

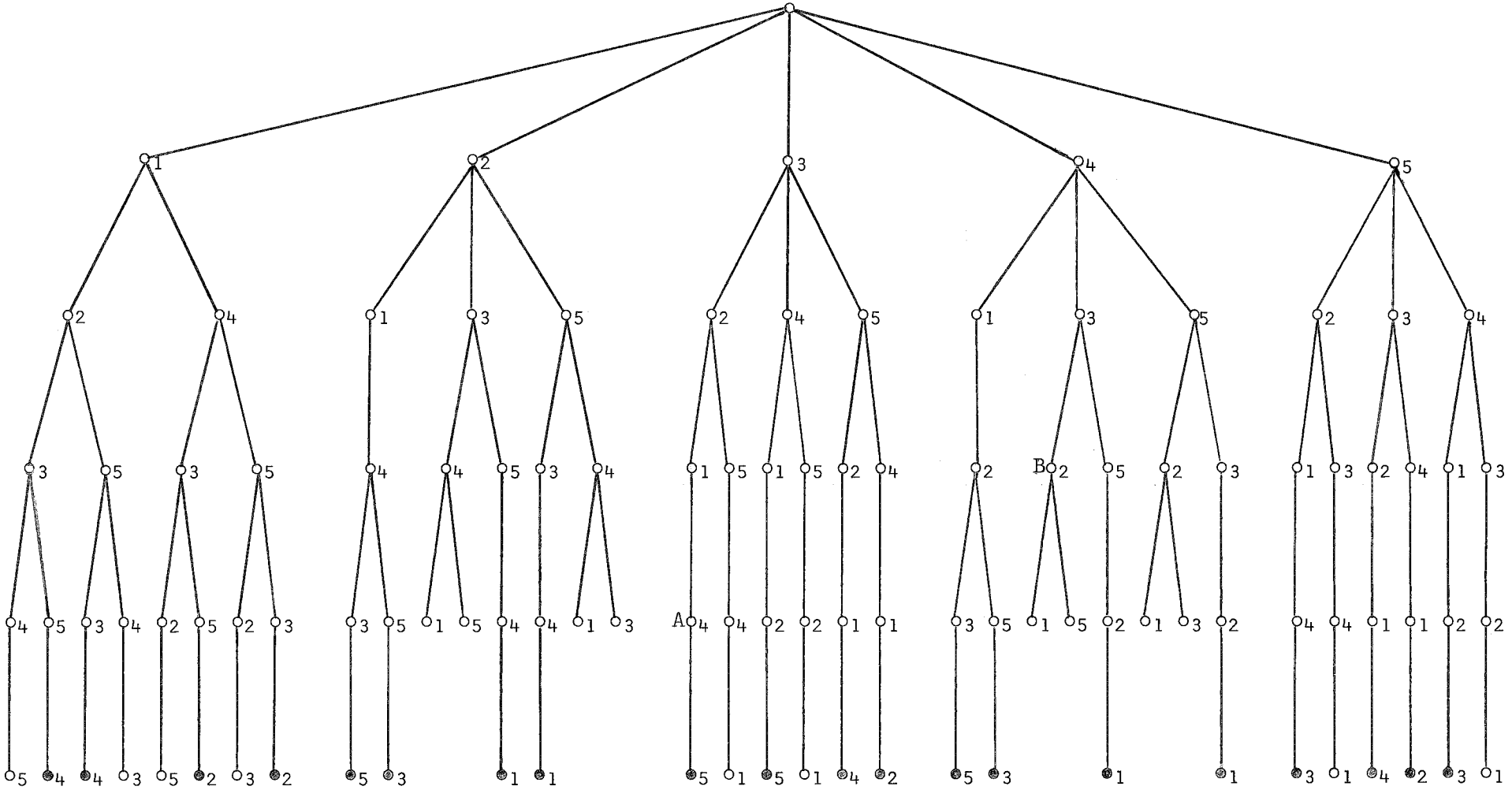


Figure 6.2

A node of the form $[v_1, \cdots, v_k, v_{k+1}]$ $(k \geq 0)$ is called a *successor* of the node $[v_1, \cdots, v_k]$. Edges of T join each node to its successors (if any). Since the edges of T are simply determined by the labels of its nodes we will normally regard T as just the set of its nodes, although we still refer to it as a tree.

Extending the successor relationship, a node v_1 of the form $[v_1, \cdots, v_k, \cdots, v_r]$ $(r > k)$ is called a *descendant* of the node $v_2 = [v_1, \cdots, v_k]$. Conversely v_2 is called an *ancestor* of v_1. If v is a node of T (we write this simply as $v \in T$), then the subset of T consisting of v and all its descendants is called the *subtree of* T *rooted at* v and is denoted $T(v)$.

If a node has no successors (and hence no descendants), it is called an *endnode* of T. If v is an endnode and $|v| = n$, then v is a *terminal node* of T. Those terminal nodes in 6.2 which satisfy P are drawn as solid circles.

6.9 Backtrack programs are notoriously sensitive to slight changes in W, and theoretical timing studies are very difficult to carry out. However, it is often possible in practice to estimate the efficiency of such a program by examining a random selection of subtrees of T. See Knuth [34] for further details.

6.10 In our analysis of program trees we shall focus our attention on the terminal nodes rather than on the solution nodes, which depend on P. In this sense the program tree is defined by the function W. In fact, we shall refer to W as a *defining function* for T. However, T may have many defining functions since it is not affected by the value of $W(v)$ when $v \notin T$.

6.11 From now on we will assume that T has at least two nodes. The *successor function* for T is the map

$$F : Q(V) \to 2^T$$

defined by

$$F(\nu) = \begin{cases} \{\mu \in Q(V) \mid \mu \text{ is a successor of } \nu\} & \nu \in T \\ \phi & \nu \notin T \end{cases}$$

It is generally more convenient to work with F rather than with W, since F and T uniquely define each other.

Let $\nu = [v_1, \cdots, v_k] \in Q(V)$, $\gamma \in S_n$. Then we write ν^γ for $[v_1{}^\gamma, \cdots, v_k{}^\gamma]$.

6.12 Theorem: *T is invariant under γ iff F commutes with γ in the sense that for any $\nu \in Q(V)$, $F(\nu^\gamma) = (F(\nu))^\gamma$.*

Proof:

(a) Suppose F commutes with γ. Let $\nu \in T$.

If ν is not an endnode of T, then $F(\nu) \neq \phi$. Hence $F(\nu^\gamma) = (F(\nu))^\gamma \neq \phi$, showing that $\nu^\gamma \in T$.

If ν is an endnode of T, then there exists $\mu \in T$ such that $\nu \in F(\mu)$. As above, $\mu^\gamma \in T$ and $F(\mu^\gamma) = (F(\mu))^\gamma$. Hence $\nu^\gamma \in T$.

(b) Suppose T is invariant under γ.

If $\nu \notin T$, then $\nu^\gamma \notin T$ and so $F(\nu) = F(\nu^\gamma) = \phi$.

If $\nu \in T$, then $\nu^\gamma \in T$. Suppose $\mu \in F(\nu)$. Then $\mu^\gamma \in T$ and so $\mu^\gamma \in F(\nu^\gamma)$.

Similarly, if $\mu^\gamma \in F(\nu^\gamma)$, then $\mu^\gamma \in T$, which shows that $\mu \in T$ and hence $\mu \in F(\nu)$.

Thus $F(\nu^\gamma) = (F(\nu))^\gamma$. □

6.13 Theorem: *Let* E *be the set of endnodes of* T. *Let* $\gamma \in S_n$. *Then* $T = T^\gamma$ *iff* $E = E^\gamma$.

Proof: By definition, $E = \{\nu \in T | F(\nu) = \phi\}$.

(a) Suppose $T = T^\gamma$. Let $\nu \in E$.

Then $\nu^\gamma \in T$ and $F(\nu^\gamma) = (F(\nu))^\gamma$ by 6.12. Hence $\nu^\gamma \in E$.

(b) Suppose $E = E^\gamma$. Let $\nu \in T$ where $\nu = [v_1, \cdots, v_k]$.

Then T has an endnode of the form $\mu = [v_1, \cdots, v_k, \cdots, v_r]$ where $k \leq r \leq n$.

Hence $\mu^\gamma = [v_1{}^\gamma, \cdots, v_k{}^\gamma, \cdots, v_r{}^\gamma] \in T$, and so $\nu^\gamma = [v_1{}^\gamma, \cdots, v_k{}^\gamma] \in T$. □

6.14 Theorem: *If* $T = T^\gamma$ *and* X *is the set of terminal nodes of* T, *then* $X = X^\gamma$.

Proof: $X = T \cap Q^{(n)}(V)$ and $|\nu^\gamma| = |\nu|$ for any $\nu \in T$, $\gamma \in S_n$. □

6.15 Theorem: *Let* $\Gamma(T) = \{\gamma \in S_n | T = T^\gamma\}$. *Then* $\Gamma(T)$ *is a group.*

Proof: Let $\gamma_1, \gamma_2 \in \Gamma(T)$. Then $T^{\gamma_1\gamma_2} = T^{\gamma_2} = T$. □

6.16 The group $\Gamma(T)$ will be called the *invariance group* of T.

For example, if T is the program tree of Figure 6.2, then $T(T)$ is the group $\{(1), (24), (35), (24)(35)\}$. In this case $T(T)$ is precisely the automorphism group of the graph G. This situation is very common and will be considered in more depth later.

Recall that $T(v)$ is the subtree of T rooted at v. The motivation for the study of $T(T)$ can be found in the following result.

6.17 Theorem: *Let* $\gamma \in T(T)$ *and* $v \in T$. *Then* $T(v^\gamma) = (T(v))^\gamma$.

Proof: Suppose $v = [v_1, \cdots, v_k]$. Then $v^\gamma = [v_1{}^\gamma, \cdots, v_k{}^\gamma] \in T$.

If μ is a descendant of v, then it has the form $\mu = [v_1, \cdots, v_k, \cdots, v_r]$ $(k < r \leq n)$. Thus $\mu^\gamma = [v_1{}^\gamma, \cdots, v_k{}^\gamma, \cdots, v_r{}^\gamma] \in T(v^\gamma)$.

Similarly, if $\mu^\gamma \in T(v^\gamma)$, then $\mu \in T(v)$ since $\gamma^{-1} \in T(T)$ by 6.15. □

6.18 We consider the consequences of 6.17. Given any subtree $T(v)$ and permutation $\gamma \in T(T)$, we can construct the subtree $T(v^\gamma)$ without the need for producing it by using the backtrack Algorithm 6.5. In particular, the terminal nodes of $T(v^\gamma)$ can be determined from those of $T(v)$.

Taking this idea a step further, let Ψ be a subgroup of $T(T)$, and let $v, \mu \in X$. Then we write $v \underset{\Psi}{\sim} \mu$ if $\mu = v^\gamma$ for some $\gamma \in \Psi$.

By 6.14 the relation $\underset{\Psi}{\sim}$ (written as $\sim$ if Ψ is understood)

is an equivalence relation on X. Consequently X can be determined from the group Ψ and any subset $R \subseteq X$ containing at least one node from each equivalence class under $\sim$. This can produce a considerable saving if $|\Psi|$ is large. A means of producing R using algorithm 6.5 will be given as soon as a few additional results are discussed.

If $\Psi \leq T(T)$ and $\nu = [v_1, \cdots, v_k] \in T$, then Ψ_ν denotes the point-wise stabiliser of $\{v_1, \cdots, v_k\}$ in Ψ.

6.19 Lemma: *Let* $\nu \in T$, $T = T(T)$. *Then* $T_\nu \leq T(T(\nu))$.

Proof: Let $\gamma \in T_\nu$ and $\mu = [v_1, \cdots, v_k, \cdots, v_r]$ $(k \leq r \leq n)$ where $\nu = [v_1, \cdots, v_k]$ and $\mu \in T(\nu)$.

Then $\mu^\gamma = [v_1^\gamma, \cdots, v_k^\gamma, v_{k+1}^\gamma, \cdots, v_r^\gamma]$

$= [v_1, \cdots, v_k, v_{k+1}^\gamma, \cdots, v_r^\gamma] \in T(\nu)$. □

Unfortunately, we do not always have equality in 6.19. For example, if ν is the node marked B in Figure 6.2, $T(T(\nu)) = \{(1), (15)\}$ but $T_\nu = \{(1)\}$.

6.20 Lemma: *Let* $\nu \in T$, $\Psi \leq T(T)$ *and let* W *be a defining function for* T. *Then* $W(\nu)$ *is a union of orbits of* Ψ_ν.

Proof: Let $\gamma \in \Psi_\nu$. Then $\nu^\gamma = \nu$. Hence $W(\nu) = W(\nu^\gamma) = (W(\nu))^\gamma$ by 6.12. □

6.21 Let $\Psi \leq T(T)$ and suppose W is a defining function for T. We define a *quotient tree* T/Ψ as the program tree given by a defining function W/Ψ constructed as follows:

Let $\nu = [v_1, \cdots, v_k] \in Q(V)$.

(1) If $\nu \notin T$ set $(W/\Psi)(\nu) = \phi$.

(2) If $\nu \in T$ then by 6.20 $W(\nu)$ is a union of orbits of Ψ_ν. Define $(W/\Psi)(\nu)$ to be any set consisting of exactly one element from each of these orbits.

The tree T/Ψ depends on the method of choosing orbit representatives of Ψ_ν and so is not uniquely defined.

6.22 For example, we take the tree T of Figure 6.2 and the group $\Psi = \{(1), (24), (35), (24)(35)\}$. Then a quotient tree T/Ψ is shown in Figure 6.3. The nodes are labelled in the same fashion as for Figure 6.2.

As indicated earlier, the value of $W(\nu)$ when $\nu \notin T$ is arbitrary and does not affect T. Since also $T/\Psi \subseteq T$ by its definition, we can construct W/Ψ from W and Ψ and so Algorithm 6.5 can be used to find T/Ψ. The example suggests that T/Ψ is considerably smaller than T and this is indeed true in the sense of the following result.

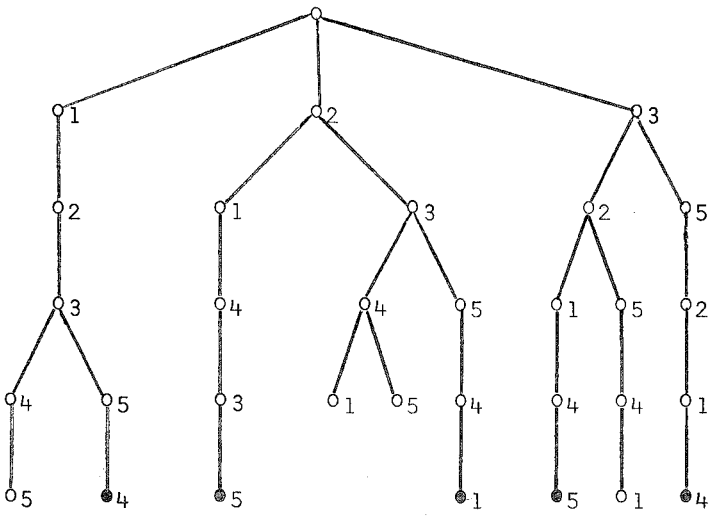


Figure 6.3

6.23 Theorem: *Let* X *and* R *be the sets of terminal nodes of* T *and* T/Ψ *respectively. Consider the equivalence classes of* X *defined in 6.18. Then* R *contains exactly one member of each equivalence class.*

Proof: $R \subseteq X$ since $T/\Psi \subseteq T$.

(1) Let $\nu = [v_1, \cdots, v_n] \in X$. Then we can construct $\mu \in R$ such that $\nu \sim \mu$ as follows:

For $0 \leq k \leq n$ let $\nu_k = [v_1, \cdots, v_k]$. Then $\nu_k \in T$. Suppose we have found, for some k, $\mu_k = [w_1, \cdots, w_k] \in T/\Psi$ and $\gamma \in \Psi$ such that $\mu_k = \nu_k{}^{\gamma}$. Now $v_{k+1} \in W(\nu_k)$ and so by 6.12, $v_{k+1}{}^{\gamma} \in W(\mu_k)$.

Consequently there is $w_{k+1} \in (W/\Psi)(\mu_k)$ and $\delta \in \Psi_{\mu_k}$ such that $w_{k+1} = v_{k+1}{}^{\gamma\delta}$. So $\mu_{k+1} \in T/\Psi$ where $\mu_{k+1} = [w_1, \cdots, w_k, w_{k+1}]$, and since $\delta \in \Psi_{\mu_k}$, $\mu_{k+1} = \nu_{k+1}{}^{\gamma\delta}$ where $\gamma\delta \in \Psi$.

Continuing this process we find that $\mu_n \sim \nu$.

(2) Suppose there are distinct elements $\nu_1, \nu_2 \in R$ and $\gamma \in \Psi$ such that $\nu_2 = \nu_1{}^{\gamma}$. Let $\nu_1 = [v_1, \cdots, v_n]$. Then ν_1 and ν_2 have a common ancestor of greatest length $\mu = [v_1, \cdots, v_k]$. Then $\gamma \in \Psi_{\mu}$. Hence, by the definition of W/Ψ, we have $v_{k+1} = v_{k+1}{}^{\gamma}$ contradicting the maximality of μ. □

6.24 Corollary: $|X| = |\Psi||R|$. □

6.25 We have shown that knowledge of a subgroup $\Psi \leq T(T)$ can be used to significantly reduce the amount of work required by the backtrack algorithm. However, we have not indicated how such a subgroup could be found. There seems to be no way of doing this in

general, except by computing the entire tree T, and this is what we are aiming to avoid. However, when the elements of the set V are the objects of a set with "suitable structure" (for example, the points of a graph, the elements of a group, or the vertices of a polyhedron) then the "automorphisms" (structure-preserving permutations) can very commonly be identified as elements of $\Gamma(T)$. So that we can avoid the difficulties in defining these ideas in a precise general fashion, we shall describe the case where V is the set of points of a graph.

In order to represent a graph in a computer, it is necessary to label the points of the graph in some manner. To take the most common situation, we are given a set of labels, normally $\{1, \cdots, n\}$ and must assign each label to a point of the graph in some arbitrary (one-to-one) fashion. The condition we require is that computation of the defining function W does not depend on the way in which this labelling is performed. Let us make this rigorous.

Suppose the computation of W is carried out by a procedure

$$\mathcal{W}: \ \underset{\sim}{G}(V) \times Q(V) \to 2^V$$

so that for $G \in \underset{\sim}{G}(V)$, $\nu \in Q(V)$, the computed value of $W(\nu)$ will be $\mathcal{W}(G, \nu)$. The procedure $\mathcal{W}$ can be said to be *independent of the labelling of* G if for $\gamma \in S_n$, $\nu \in Q(V)$ we have

$$\mathcal{W}(G^\gamma, \nu^\gamma) = (\mathcal{W}(G, \nu))^\gamma. \tag{6.26}$$

6.27 Theorem: *If $\mathcal{W}$ is independent of the labelling of* G, *then* $\Gamma(G) \leq \Gamma(T)$.

Proof: If $\gamma \in \Gamma(G)$, then $G^\gamma = G$. Hence for any $\nu \in Q(V)$, 6.26

becomes

$$\mathcal{W}(G, \nu^\gamma) = (\mathcal{W}(G, \nu))^\gamma,$$

or equivalently,

$$W(\nu^\gamma) = (W(\nu))^\gamma.$$

Therefore, if F is the successor function for T, then $F(\nu^\gamma) = (F(\nu))^\gamma$ and so $\gamma \in T(T)$ by 6.12. □

In practice, it is usually quite easy to decide whether $\mathcal{W}$ is independent of the labelling of G. Roughly speaking, this will be the case if $\mathcal{W}$ treats the labels as objects without any ordering and makes no arbitrary choices. However, there is another method of showing $\Gamma(G) \leq T(T)$ which is often easier to apply. This method consists of identifying the endnodes of T and using Theorem 6.13. To illustrate this we take our former example and the function W defined in 6.6. If γ is an automorphism of G and $[v_1, \cdots, v_k]$ is an endnode of T, then so is $[v_1^\gamma, \cdots, v_k^\gamma]$ since γ preserves adjacency. Effectively, we need only verify 6.26 for those ν where $\mathcal{W}(G, \nu) = \phi$.

6.28 Although our development so far has been quite straightforward, these ideas have received only scant attention. This is perhaps partly explained by the following practical difficulties:

(1) Computation of $\Gamma(G)$ is required. Although many known algorithms are capable of computing $\Gamma(G)$, they invariably generate each element of $\Gamma(G)$ individually. When $|\Gamma(G)|$ is large this may take impossibly long. In any case, finding $\Gamma(G)$ may take longer than using the original version of the backtrack algorithm.

(2) Once $\Gamma(G)$ has been computed we have the problem of storing

it in the computer. The methods described in Chapter 4 may be used, but these do not seem to be widely known.

(3) The evaluation of the defining function W/Γ requires the orbits of the stabiliser Γ_ν for a possibly large number of nodes ν. Unfortunately, in the notation of 4.4, there seems to be no easy way of converting a set $\{\gamma_i^{(k)}\}$ of coset representatives corresponding to a sequence $[v_1, \cdots, v_{r+1}]$ to a set corresponding to another sequence $[w_1, \cdots, w_{r+1}]$. The constant need to compute the orbits of Γ_ν may take more time than it saves, unless the computation of $W(\nu)$takes a similar amount of time.

6.29 In order to avoid these problems we can use various compromises. For example,

(1) We can use only a small subgroup of $\Gamma(G)$. The result 6.24 indicates that even the subgroup Ψ generated by a single element of $\Gamma(G)$ may considerably reduce the size of the program tree. In this case the computation of Ψ_ν is trivial.

(2) We can restrict our attention to subgroups of $\Gamma(G)$ of special type. In Sections 6.30-6.33 we shall consider the subgroup of $\Gamma(G)$ generated by its transpositions. This method will of course be useless if $\Gamma(G)$ has no transpositions.

(3) We can use a more sophisticated means of reducing the size of the program tree. Several such methods will be presented in Chapter 7.

6.30 Lemma: *Let* $G \in \underset{\sim}{G}(V)$ *and* $v, w \in V$. *Then the transposition* (vw) *is in* $\Gamma(G)$ *iff* v *is adjacent to the same points in* $V\backslash\{v, w\}$ *as is* w.

Proof: Trivial. □

This result shows that the transpositions in $\Gamma(G)$ can be easily found. The next three results show how the subgroup they generate may be handled.

6.31 Lemma: [54] *Let* V_1 *be a subset of* V. *Then if* $Z \leq S(V_1)$ *is a set of transpositions,* Z *generates* $S(V_1)$ *iff* $\theta_Z = V_1$. □

6.32 Lemma: *If* $\Psi \leq S(V)$, *then* Ψ *is generated by transpositions iff* $\Psi = S(V)_\pi$ *where* $\pi = \theta_\Psi$.

Proof: Suppose $\pi = \{C_1|C_2|\cdots|C_k\}$. Then by applying 6.31 to each cell C_i we see that

$$\Psi = S(C_1) \oplus \cdots \oplus S(C_k).$$ □

6.33 Lemma: *If* $\Psi \leq S(V)$ *is generated by transpositions, and* $\pi \in \Pi(V)$, *then* $\theta_\Lambda = \theta_\Psi \wedge \pi$ *where* $\Lambda = \Psi_\pi$.

Proof: Clearly $\theta_\Lambda \leq \theta_\Gamma$ (4.14) and $\theta_\Lambda \leq \pi$ (trivial).

Hence $\theta_\Lambda \leq \theta_\Gamma \wedge \pi$.

But $\theta_\Lambda \geq \theta_\Gamma \wedge \pi$ by 6.32. □

Lemmas 6.32 and 6.33 show that only the partition θ_Ψ is required in order to evaluate W/Ψ. If $\nu \in Q(V)$ the orbits of Ψ_ν which lie in $W(\nu)$ are simply the non-null sets of the form $W(\nu) \cap C_i$ where $\theta_\Psi = \{C_1|\cdots|C_k\}$. Thus the quotient tree T/Ψ can be generated very easily. In the context of

graph isomorphism this idea was first used by Morgan [46] who considered the canonical labelling of chemical compounds. A more general treatment was given later by Steen [69].

CHAPTER SEVEN

BACKTRACK PROGRAMMING - II

7.1 We are now in a position to present a number of techniques by which we can reduce the size of a program tree T without prior explicit knowledge of T(T). In order for these techniques to work we require a means for "recognising" some subgroup Ψ of T(T), in the sense that, given $\gamma \in S_n$, we can decide whether or not $\gamma \in \Psi$. For example, if we are working with a graph G and $\Gamma(G) \leq T(T)$, then by permuting the adjacency matrix of G we can tell whether or not $\gamma \in \Gamma(G)$. Clearly any subgroup of T(T) is "recognisable" in principle, but our techniques will not be practically useful unless the recognition can be performed with reasonable efficiency. Throughout this chapter, we continue the notation of Chapter Six, and assume that $\Psi \leq T(T)$. Except as indicated in 7.28, all of this chapter is original.

Let T be the program tree with defining function W and having successor function F. Let X be the set of terminal nodes of T; for convenience we assume that X is not empty. The elements of X will be assumed to be in the order in which they are produced; for example, from left to right in Figure 6.2. Hence, for example, we can talk of $\tau_1 \in X$ being *earlier* than $\tau_2 \in X$. Similarly, if $\nu_1, \nu_2 \in T$ we can say that $T(\nu_1)$ is *earlier* than $T(\nu_2)$ if every terminal node of $T(\nu_1)$ is earlier than those of $T(\nu_2)$. Following 6.18 we denote $\tau_1 \sim \tau_2$ if for some $\gamma \in \Psi$, $\tau_2 = \tau_1^{\gamma}$. Such terminal nodes will be called *equivalent* (under Ψ). The earliest terminal nodes in each equivalence class will be called *identity nodes* and denoted $\{e_1, \cdots, e_r\}$ in the order in which they are produced, where $|X| = r|\Psi|$.

Let $\tau_1, \tau_2 \in X$ where $\tau_1 \neq \tau_2$. Suppose $\tau_1 = [v_1, \cdots, v_n]$ and $\tau_2 = [w_1, \cdots, w_n]$ where $v_i = w_i$ $(0 \leq i \leq k)$ and $v_{k+1} \neq w_{k+1}$. Then we denote $\tau_1 - \tau_2 = [v_1, \cdots, v_k, v_{k+1}]$ and $\tau_2 - \tau_1 = [w_1, \cdots, w_k, w_{k+1}] = [v_1, \cdots, v_k, w_{k+1}]$. For example, if $\tau_1 = [1, 2, 3, 5, 4]$ and $\tau_2 = [1, 2, 5, 3, 4]$, $\tau_1 - \tau_2 = [1, 2, 3]$ and $\tau_2 - \tau_1 = [1, 2, 5]$. Since τ_1 and τ_2 are descendants of $\tau_1 - \tau_2$ and $\tau_2 - \tau_1$ respectively, $\tau_1 - \tau_2$ and $\tau_2 - \tau_1$ are both in T.

7.2 Lemma: *Let* $T(\nu_1)$ *and* $T(\nu_2)$ *be subtrees of* T *where* $\nu_2 = \nu_1^{\gamma}$ *for some* $\gamma \in \Psi$, *but* $\nu_2 \neq \nu_1$. *Then if* $T(\nu_1)$ *is earlier than* $T(\nu_2)$, $T(\nu_2)$ *contains no identity nodes.*

Proof: By 6.17, $T(\nu_2) = (T(\nu_1))^{\gamma}$. Therefore, if $T(\nu_2)$ contains an identity node e, $e^{\gamma^{-1}}$ is earlier than e, which is a contradiction. □

Suppose that at some stage during the execution of Algorithm 6.5 we have encountered the identity nodes $\{e_1, \cdots, e_s\}$ and now find the terminal node τ. There are two possibilities:

(1) τ is a new identity node.

(2) $\tau \sim e_i$ for some i $(1 \leq i \leq s)$. Suppose $\tau = e_i^{\gamma}$ where $\gamma \in \Psi$. Then, if $e_i - \tau = [v_1, \cdots, v_k, v_{k+1}]$ we have

$$\tau - e_i = [v_1, \cdots, v_k, v_{k+1}^{\gamma}].$$

Hence $$\tau - e_i = (e_i - \tau)^{\gamma}$$

and so $$T(\tau - e_i) = (T(e_i - \tau))^{\gamma}.$$

Since also $T(e_i - \tau)$ is earlier than $T(\tau - e_i)$ we conclude from 7.2 that $T(\tau - e_i)$ contains no identity nodes. Thus we can

remove $T(\tau - e_i)$ from the tree without losing identity nodes.

These ideas lead us to the following simple algorithm, which is modelled on 6.5.

7.3 Algorithm: Find the identity nodes of T.

(1) Set $k := 0$; $s := 0$.

(2) Set $U_k := W([v_1, \cdots, v_k])$.

(3) If $U_k = \phi$ go to step (9).

(4) Choose and delete any element v_{k+1} from U_k. Set $k := k + 1$.

(5) If $k < n$ go to step (2).

(6) We have found a terminal node $\tau = [v_1, \cdots, v_n]$. If $\tau \sim e_j$ for some j $(1 \leq j \leq s)$ go to step (8).

(7) Set $s := s + 1$; $e_s := \tau$. Go to step (9).

(8) Set $k := |\tau - e_j|$.

(9) Set $k := k - 1$. If $k \geq 0$ go to step (3); otherwise stop.

7.4 We now apply Algorithm 7.3 to the example of 6.6, taking $\Psi = \Gamma(G)$. For the first two terminal nodes of T, Algorithm 7.3 behaves the same as Algorithm 6.5 and so we will not repeat this part. Instead, we take up the workings where we left off in 6.7. At this stage we have found two non-equivalent terminal nodes. The various symbols have values as follows:

$e_1 = [1, 2, 3, 4, 5]$

$e_2 = [1, 2, 3, 5, 4]$

$v_1 = 1, \quad U_0 = \{2, 3, 4, 5\}$

$v_2 = 2, \quad U_1 = \{4\}$

$v_3 = 3, \quad U_2 = \{5\}$

$v_4 = 5, \quad U_3 = \phi$

$v_5 = 4, \quad U_4 = \phi$

$k = 5, s = 2.$

(9) $k = 4.$

(3) $U_4 = \phi$ so go to (9).

(9) $k = 3.$

(3) $U_3 = \phi$ so go to (9).

(9) $k = 2.$

(4) $v_3 = 5, U_2 = \phi, k = 3.$

(5) $k < n$ so go to (2).

(2) $U_3 = \{3, 4\}.$

(3) $U_3 \neq \phi.$

(4) $v_4 = 3, U_3 = \{4\}, k = 4.$

(5) $k < n$ so go to (2).

(2) $U_4 = \{4\}.$

(3) $U_4 \neq \phi.$

(4) $v_5 = 4, U_4 = \phi, k = 5.$

(5) $k = n.$

(6) $\tau = [1, 2, 5, 3, 4]; \tau \sim e_2;$ go to (8).

(8) $k = |[1, 2, 5]| = 3.$

(9) $k = 2.$

etc.

Continuing this process we obtain the program tree shown in Figure 7.1. Comparing this with Figure 6.2 we see that the number of terminal nodes has been reduced from 28 to 15. The terminal nodes in Figure 7.1 are labelled according to their equivalence classes and the automorphisms $\alpha = (24)$ and $\beta = (35)$.

7.5 We have shown that Algorithm 7.3 produces the full set of identity nodes $\{e_1, \cdots, e_r\}$. These can be thought of as the terminal nodes of some quotient tree T/Ψ. In many applications the set $\{e_1, \cdots, e_r\}$, since it represents all terminal nodes not equivalent under Ψ, will be all that is required. However, if we need the entire set of terminal nodes of T, we first need to find Ψ. It turns out that Ψ can be constructed quite simply from those elements of Ψ which are encountered during the execution of the algorithm.

Let T and T_1 be respectively the program trees produced by Algorithms 6.5 and 7.3.

Suppose that during the execution of 7.3 we have found an identity node e_j and a terminal node τ such that $\tau \neq e_j$ but $\tau = e_j^{\gamma}$ for some $\gamma \in \Psi$. Then we say that $\tau - e_j$ is *absorbed onto* $e_j - \tau$ *by* γ. In Figure 7.1 such absorptions are indicated by dashed arrows.

In our analysis of T_1 we are assuming that the orders of choosing the v_{k+1} from U_k at step (4) of Algorithms 6.5 and 7.3 are the same.

7.6 Lemma: *Let* $e_i = [v_1, \cdots, v_n]$ *be an identity node of* T. *Then any node* v *of* T *of the form* $[v_1, \cdots, v_k, w]$ $(0 \leq k < n)$ *will also be in* T_1.

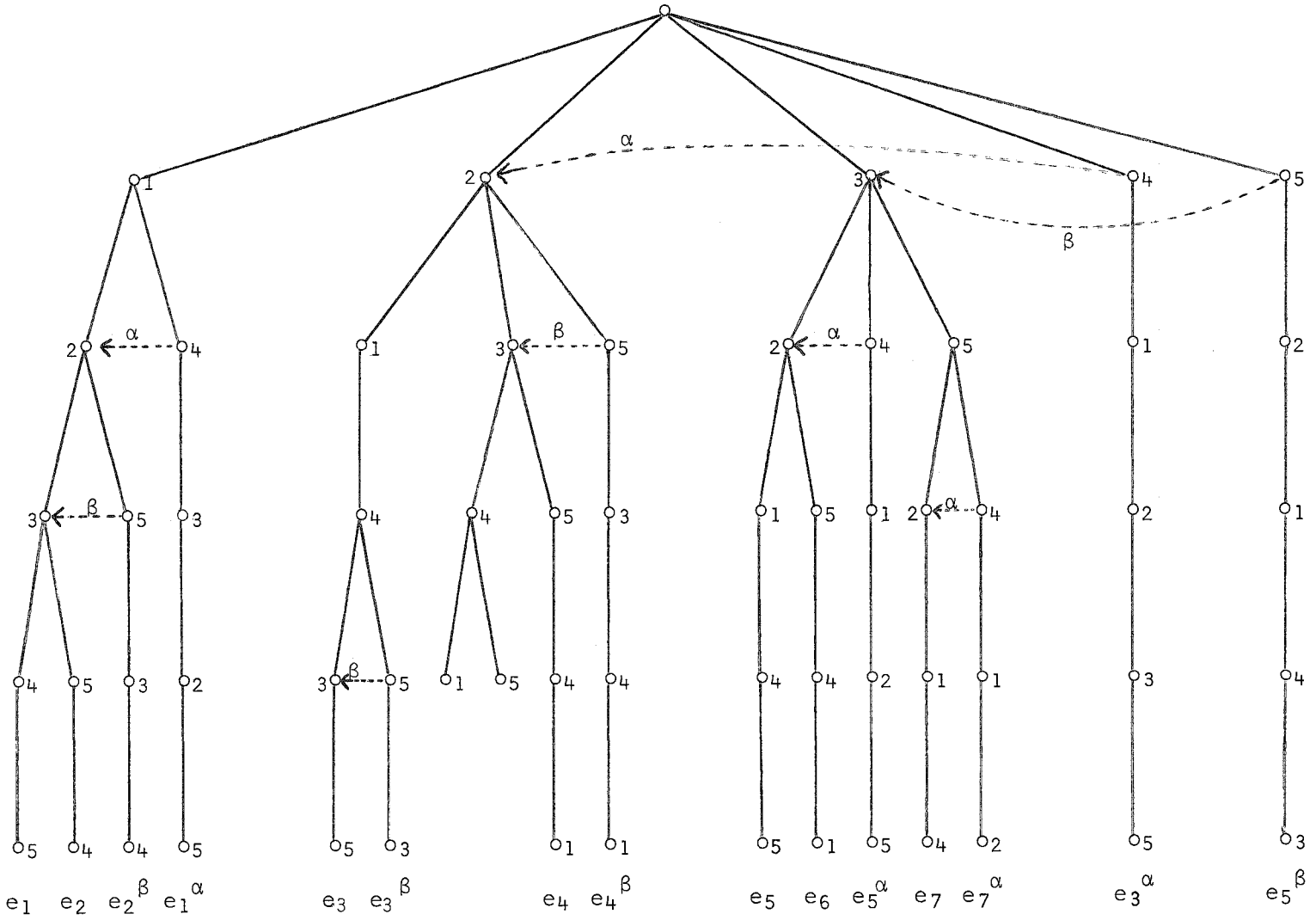


Figure 7.1

α = (2 4), β = (3 5)

Proof: If ν is not in T_1, then some ancestor μ_2 of ν must have been absorbed by an element γ of Ψ onto a node μ_1. But then $e_i^{\gamma^{-1}}$ is earlier than e_i, contradicting the assertion that e_i is an identity node. □

7.7 Corollary: *If* $\tau \in X$, *then* $\tau - e_i \in T_1$.

Proof: $\tau - e_i$ is of the form required by 7.6. □

7.8 Lemma: *Let* $e_i = [v_1, \cdots, v_n]$ *be an identity node of* T. *Let*

$$\nu_1 = [v_1, \cdots, v_k, v_{k+1}] \text{ and}$$

$$\nu_2 = [v_1, \cdots, v_k, w] \text{ where } \nu_2 \in T \text{ and } \nu_2 = \nu_1^{\gamma}$$

for some $\gamma \in \Psi$, *but* $\nu_2 \neq \nu_1$. *Then* ν_2 *will be absorbed onto* ν_1 *(but not necessarily by* γ*).*

Proof: Let τ be the first terminal node of $T(\nu_2)$. Then $\nu_1 = e_i - \tau$ and $\nu_2 = \tau - e_i$. Since $T(\nu_1)$ is earlier than $T(\nu_2)$, τ is not an identity node, by 7.2. Hence there is an identity node e_j and an element $\delta \in \Psi$ such that $\tau = e_j^{\delta}$.

Now $\tau - e_j$ and ν_2 are both ancestors of τ.

(1) Suppose $\tau - e_j$ is an ancestor of ν_2.

Then $e_i \in T(\tau - e_j)$ since $e_i \in T(\nu_1)$ and $\tau - e_j$ is an ancestor of ν_1.

But $\tau - e_j = (e_j - \tau)^{\delta}$ and $T(e_j - \tau)$ is earlier than $T(\tau - e_j)$, which contradicts 7.2.

(2) Suppose ν_2 is an ancestor of $\tau - e_j$.

Then $e_j \in T(\nu_2)$ since $e_j \in T(e_j - \tau)$ and ν_2 is an ancestor of $e_j - \tau$.

But $\nu_2 = \nu_1^{\gamma}$ and $T(\nu_1)$ is earlier than $T(\nu_2)$, which again contradicts 7.2.

Hence we must have $\nu_2 = \tau - e_j$. Let $\nu_3 = \nu_2^{\delta^{-1}}$. Then $\nu_3 = \nu_1^{\gamma\delta^{-1}}$. But $e_i \in T(\nu_1)$ and $e_j \in T(\nu_3)$ and so $\nu_1 = \nu_3$ by 7.2. Therefore $\nu_1 = e_j - \tau$ and $\nu_2 = \tau - e_j$, and so ν_2 will be absorbed onto ν_1 by δ. □

Let $e_j = [v_1, \cdots, v_n]$ be an identity node of T. For $0 \le k \le n$ define $\nu_k = [v_1, \cdots, v_k]$ and $\Psi^{(k)} = \Psi_{\nu_k}$.

7.9 Theorem: *For* $0 \le k < n$ *(following 4.4) we have the disjoint union*

$$\Psi^{(k)} = \bigcup_{i=1}^{s_k} \Psi^{(k+1)}\gamma_i^{(k)}$$

where $\gamma_1^{(k)} = (1)$ *and* $\{\gamma_2^{(k)}, \cdots, \gamma_{s_k}^{(k)}\}$ *are the elements of* Ψ *by which nodes of* T *are absorbed onto* ν_{k+1}.

Proof: Let the orbit of $\Psi^{(k)}$ which contains v_{k+1} be $Z = \{w_1, \cdots, w_{s_k}\}$ where $w_1 = v_{k+1}$.

By 6.20 $Z \subseteq W(\nu_k)$, and so by 7.6, $\mu_i \in T_1$ where $\mu_i = [v_1, \cdots, v_k, w_i]$ $(1 \le i \le s_k)$. Note that $\mu_1 = \nu_{k+1}$ and consider μ_i, where $2 \le i \le s_k$.

By 7.8, μ_i will be absorbed onto μ_1 by an element $\gamma_i^{(k)}$ of Ψ. Since ν_k is a common ancestor of μ_1 and μ_2, $\gamma_i^{(k)} \in \Psi_{\nu_k} = \Psi^{(k)}$. Furthermore, $\gamma_i^{(k)}$ maps w_1 onto w_i. The theorem follows from 4.2. □

7.10 Corollary: *For any* $0 \leq h < n$, $\Psi^{(h)}$ *is generated by the set*

$$\Omega_h = \{\gamma_i^{(k)} | h \leq k < n,\ 1 \leq i \leq s_k\}.$$

In particular, Ω_0 *generates* Ψ.

Proof: By 4.5. □

Theorem 7.9 shows that in order to find Ψ we must only look at those nodes which are absorbed onto ancestors of a single fixed identity node -- for example, the first terminal node e_1.

For the tree of Figure 7.1, we find

$$\begin{aligned}
\Psi = \Psi^{(0)} &= \Psi^{(1)}, \\
\Psi^{(1)} &= \Psi^{(2)} \cup \Psi^{(2)}(2\ 4), \\
\Psi^{(2)} &= \Psi^{(3)} \cup \Psi^{(3)}(3\ 5), \\
\Psi^{(3)} &= \Psi^{(4)}, \\
\Psi^{(4)} &= \Psi^{(5)} = \{(1)\}.
\end{aligned}$$

Hence $\Psi = \langle (2\ 4), (3\ 5) \rangle$ as expected.

Theorem 7.9 also enables us to find a bound for the number of terminal nodes of T_1. Recall that the terminal nodes of T are the set X where $|X| = r|\Psi|$.

7.11 Theorem: T_1 *has* t *terminal nodes, where* $t \leq r(\binom{n}{2} + 1)$.

Proof: Let e be an identity node of T. By 7.9 the number of nodes absorbed onto ancestors of e is

$$\sum_{k=0}^{n-1} (s_k - 1).$$

But $s_k \leq n - k$ for $0 \leq k \leq n - 1$, and so

$$\sum_{k=0}^{n-1} (s_k - 1) \leq \sum_{k=0}^{n-1} (n - k - 1) = \binom{n}{2}.$$

Therefore the number of non-identity terminal nodes associated with each identity node in this way is bounded above by $\binom{n}{2}$. The theorem follows immediately. □

The bound of 7.11 is realized only when $\Psi = S_n$ and is generally too large. Since $|\Psi|$ can be as large as n! the work saved by using 7.3 instead of 6.5 can be enormous.

In Theorem 4.8 we showed that the set Ω_0 can be reduced to a set Y' of at most n - p generators of Ψ, where Ψ has p orbits. Hence we can find such a generating set by producing Ω_0 via Algorithm 7.3 and then applying Algorithm 4.9. However a closer look at the ideas behind 7.3 reveals a way in which such a set can be produced directly.

7.12 Before proceeding further we shall establish the following conventions. It has been assumed that $V = \{1, \cdots, n\}$. If $w_1, w_2 \in V$, then by $w_1 < w_2$ we simply mean that w_1 is smaller than w_2 numerically. Furthermore, we shall assume that when required to choose an arbitrary element from a subset of V (for example, the set U_k at step (4) of Algorithm 6.5 or 7.3) we shall choose the numerically smallest element. This convention has already been adhered to in our examples. The following result is now obvious.

7.13 Lemma: *Let* $\nu_1, \nu_2 \in T$ *where* $\nu_1 = [v_1, \cdots, v_k, w_1]$, $\nu_2 = [v_1, \cdots, v_k, w_2]$ *and* $w_1 < w_2$. *Then* $T(\nu_1)$ *is earlier than* $T(\nu_2)$. □

7.14 Let $e_j = [v_1, \cdots, v_n]$ be an identity node of T and suppose $0 \leq q < n$.

For $0 \leq k \leq h$ define $\nu_k = [v_1, \cdots, v_k]$ and $\Psi^{(k)} = \Psi_{\nu_k}$.

Let $\{\gamma_1, \cdots, \gamma_m\}$ be a set of elements of Ψ by which nodes of T are absorbed onto nodes ν_k where $k > q$.

Then $\gamma_i \in \Psi^{(q)}$ $(1 \leq i \leq m)$. Therefore $\Lambda \leq \Psi^{(q)}$, where $\Lambda = \langle \gamma_1, \cdots, \gamma_m \rangle$.

By 6.20, $W(\nu_q)$ is a union of orbits of Λ.

Let $\pi = \theta_\Psi = \theta_{\gamma_1} \vee \cdots \vee \theta_{\gamma_m}$ by 4.14.

Now if $w_1 < w_2$ where $w_1, w_2 \in W(\nu_q)$ and $w_1 \overset{\sim}{\pi} w_2$ then for some $\gamma \in \Psi$, $w_2 = w_1{}^\gamma$.

Hence $\mu_2 = \mu_1{}^\gamma$ where $\mu_1 = [v_1, \cdots, v_q, w_1]$ and

$$\mu_2 = [v_1, \cdots, v_q, w_2].$$

Therefore $T(\mu_2) = (T(\mu_1))^\gamma$ by 6.17 and so $T(\mu_2)$ contains no identity nodes, by 7.2 and 7.13.

7.15 To implement these ideas, additional data items are required. Upon creating a node $\nu = [v_1, \cdots, v_k]$ we compute $W(\nu)$ and create a partition $\pi_\nu \in \Pi(W(\nu))$. Initially, π_ν is set equal to the discrete partition of $W(\nu)$. Thereafter, whenever we encounter an element $\gamma \in \Psi$ by which a node is absorbed onto a descendant of ν we set $\pi_\nu := \pi_\nu \bar{\vee} \theta_\gamma$, where $\bar{\vee}$ denotes the generalized join operation introduced in 3.4. This operation can be performed by Algorithm 3.6. At any stage during the execution of the following algorithm, we require partitions only for the current node and its ancestors

(excepting that we do not need a partition for a terminal node) and so no more than n partitions need to be stored at one time.

For convenience we shall assume that the cells of a partition π_ν are stored so that if an element of a cell C_1 is smaller than every element of another cell C_2, C_1 is stored before C_2. The structure of Algorithm 3.6 ensures that if π_ν is in this form, then $\pi_\nu \bar{\vee} \theta_\gamma$ will be also, irrespective of the order of the cells of θ_γ. In the following algorithm a cell of π_ν is regarded as having been *chosen* if any element of the cell has been chosen. Our conventions ensure that the chosen cells of π_ν are always stored before those which have not been chosen.

7.16 Algorithm: Find the identity nodes of T.

(1) Set k := 0; s := 0.

(2) Compute $Z := W([v_1, \cdots, v_k])$. If $Z = \phi$ go to step (9).

(3) Set π_k := discrete partition of Z.

(4) Set C := first cell of π_k not yet chosen;
v_{k+1} := smallest point in C;
k := k + 1.

(5) If k < n go to step (2).

(6) We have found a terminal node $\tau = [v_1, \cdots, v_n]$.
If $\tau \sim e_j$ for some j $(1 \leq j \leq s)$ go to step (8).

(7) Set s := s + 1; $e_s := \tau$. Go to step (9).

(8) Compute γ such that $\tau = e_j^\gamma$. Set $k := |\tau - e_j|$.
For $0 \leq i < k$ set $\pi_i := \pi_i \bar{\vee} \theta_\gamma$.

(9) If $k = 0$ stop.

Set $k := k - 1$.

(10) If all cells of π_k have been chosen go to step (9); otherwise go to step (4).

7.17 We again consider the example of 6.6. For brevity we only include those steps of the algorithm where variables change value.

(1) $k = 0$, $s = 0$.

(2) $Z = \{1, 2, 3, 4, 5\}$.

(3) $\pi_0 = \{1|2|3|4|5\}$.

(4) $C = \{1\}$, $v_1 = 1$, $k = 1$.

(2) $Z = \{2, 4\}$.

(3) $\pi_1 = \{2|4\}$.

(4) $C = \{2\}$, $v_2 = 2$, $k = 2$.

(2) $Z = \{3, 5\}$.

(3) $\pi_2 = \{3|5\}$.

(4) $C = \{3\}$, $v_3 = 3$, $k = 3$.

(2) $Z = \{4, 5\}$.

(3) $\pi_3 = \{4|5\}$.

(4) $C = \{4\}$, $v_4 = 4$, $k = 4$.

(2) $Z = \{5\}$.

(3) $\pi_4 = \{5\}$.

(4) $C = \{5\}$, $v_5 = 5$, $k = 5$.

(6) $\tau = [1, 2, 3, 4, 5]$ - an identity node.

(7) $s = 1$, $e_1 = [1, 2, 3, 4, 5]$.

(9) $k = 4$.

(9) $k = 3$.

(4) $C = \{5\}$, $v_4 = 5$, $k = 4$.

(2) $Z = \{4\}$.

(3) $\pi_4 = \{4\}$.

(4) $C = \{4\}$, $v_5 = 4$, $k = 5$.

(6) $\tau = [1, 2, 3, 5, 4]$ - an identity node.

(7) $s = 2$, $e_2 = [1, 2, 3, 5, 4]$.

(9) $k = 4$.

(9) $k = 3$.

(9) $k = 2$.

(4) $C = \{5\}$, $v_3 = 5$, $k = 3$.

(2) $Z = \{3, 4\}$.

(3) $\pi_3 = \{3|4\}$.

(4) $C = \{3\}$, $v_4 = 3$, $k = 4$.

(2) $Z = \{4\}$.

(3) $\pi_4 = \{4\}$.

(4) $C = \{4\}$, $v_5 = 4$, $k = 5$.

(6) $\tau = [1, 2, 5, 3, 4]$ - equivalent to e_2.

(8) $\gamma = (3\ 5)$, $\theta_\gamma = \{1|2|3, 5|4\}$.

$k = 3$.

$\pi_0 = \{1|2|3, 5|4\}$.

$\pi_1 = \{2|4\}$.

$\pi_2 = \{3, 5\}$.

(9) $k = 2$.

(9) $k = 1$.

(4) $C = \{4\}$.

•
•
•

Continuing this process we obtain the program tree shown in Figure 7.2. The labelling is the same as in Figure 7.1.

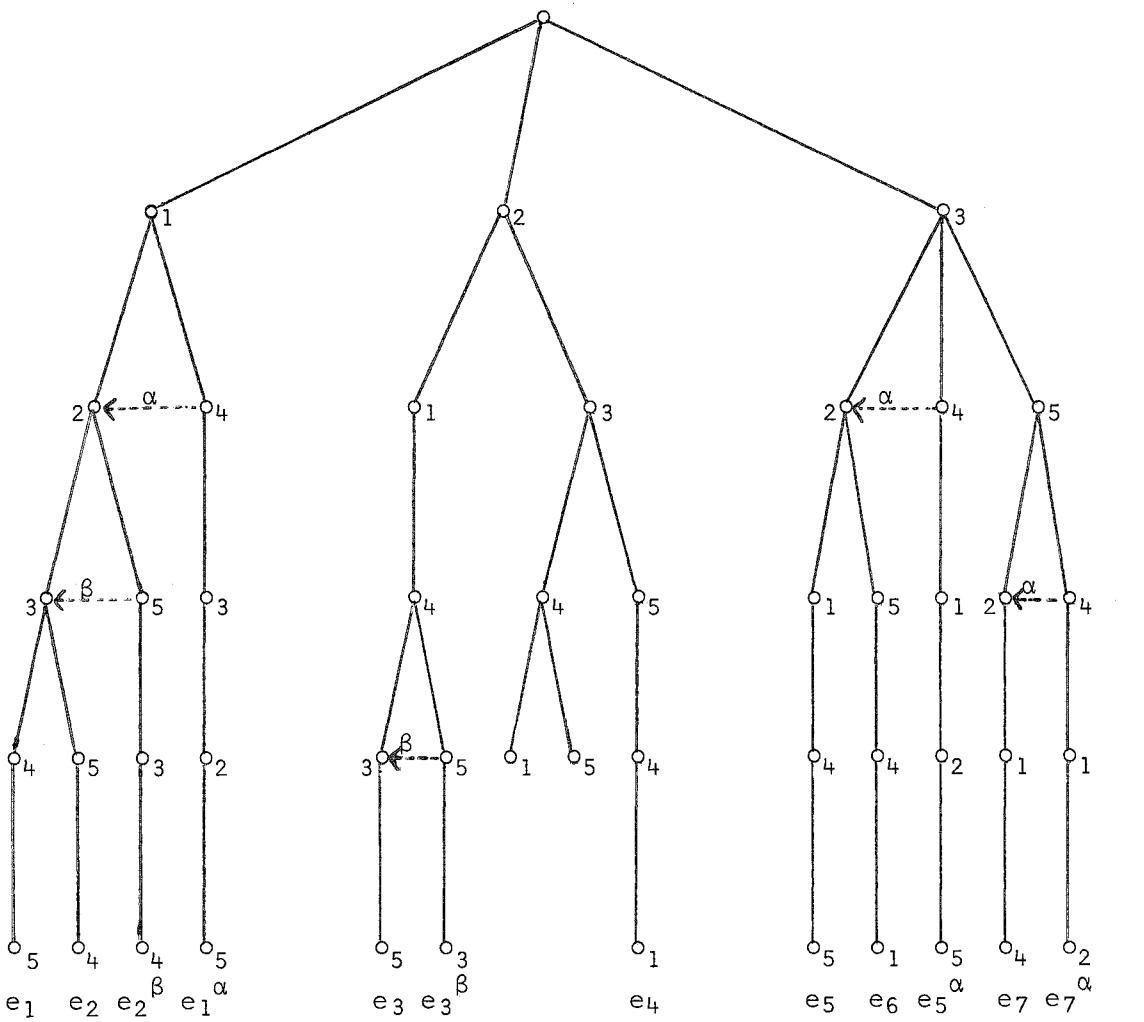
As before, if at step (8) of Algorithm 7.16, we have $\tau = e_j^\gamma$ for an identity node e_j and a terminal node τ we say that $\tau - e_j$ is *absorbed onto* $e_j - \tau$ *by* γ.

We denote by T, T_1, T_2 the program trees produced by Algorithms 6.5, 7.3 and 7.16 respectively. We have shown that both T_1 and T_2 contain the identity nodes of T.

7.18 Suppose that at step (8) of Algorithm 7.16 we have $\tau = e_j^\gamma$ where e_j is an identity node, τ a terminal node and $\gamma \in \Psi$.

Let $\tau = [v_1, \cdots, v_n]$, and $0 \leq i < |\tau - e_j|$.

Then in step (8) we set $\pi_i := \pi_i \bar{\vee} \theta_\gamma$. Suppose for some node $\nu = [v_1, \cdots, v_i, w]$ of T_2 this operation causes the cell of π_i containing w to be increased in size. Then we say that γ is *active at* ν.



α = (2 4), β = (3 5)

Figure 7.2

7.19 Lemma: *γ is active at $e_j - \tau$.*

Proof: $e_j - \tau$ is in T_2 because it is an ancestor of e_j. Also $e_j - \tau$ is clearly of the form $[v_1, \cdots, v_i, w]$ where $i = |\tau - e_j| - 1$. Then the operation $\pi_i := \pi_i \bar{v} \theta_\gamma$ brings v_{i+1} into the same cell as w. □

7.20 Theorem: *Let $e_j = [v_1, \cdots, v_n]$ be an identity node of T. Let Y be the set of elements of Ψ found by 7.16 which are active at ancestors of e_j. Then Y generates Ψ and $|Y| \leq n - p$ where Ψ has p orbits.*

Proof: We verify that Y satisfies the requirements of Theorem 4.7 for h = 0.

For $0 \leq k \leq n$, define $\nu_k = [v_1, \cdots, v_k]$,

$$\psi^{(k)} = \Psi_{\nu_k}.$$

Consider the partition π_k when the subtree $T_2(\nu_k)$ has been completely generated by the algorithm (say at step (9)).

Let Z be the orbit of $\psi^{(k)}$ containing v_{k+1}. Then $Z \subseteq W(\nu_k)$ by 6.20.

Also, Z is a union of cells of π_k, since the cells of π_k are orbits of some group generated by elements of $\psi^{(k)}$.

Suppose C_1 and C_2 are distinct cells of π_k contained in Z, where $v_{k+1} \in C_1$. Let w be the smallest element of C_2. Then w must sometime have been chosen at step (4). But this would have resulted in w being absorbed onto some element of C_1 (the proof is like that of 7.8) and C_2 and C_1 will have been merged (7.19).

Hence Z is a cell of π_k. But since $v_{k+1} \in Z$, π_k is an orbit of the group generated by elements active at v_q for each $q > k$. Therefore, the set Y satisfies the conditions of 4.7 for $h = 0$.

Hence $\langle Y \rangle = \Psi$.

Now let $Y = \{\gamma_1, \cdots, \gamma_t\}$ in the order these elements are found. For $0 \leq \ell \leq t$ define

$$\pi^{(\ell)} = \theta_{\gamma_1} \vee \cdots \vee \theta_{\gamma_\ell}.$$

Then since each γ_i is active at some ancestor of e_j, $\pi^{(\ell+1)}$ is always strictly finer than $\pi^{(\ell)}$ $(0 \leq \ell < t)$. But $\pi^{(0)}$ and $\pi^{(t)}$ have n and p cells respectively, and so $t \leq n - p$. □

7.21 Corollary: *For* $0 \leq k \leq n$, $\Psi^{(k)} = \langle Y \cap \Psi^{(k)} \rangle$.

Proof: Immediate from 4.7. □

7.22 Theorem: T_2 *has at most* $r(n - p + 1)$ *terminal nodes, where* $|X| = r|\Psi|$ *and* Ψ *has* p *orbits.*

Proof: Let τ be a terminal node which is not an identity node. Then for some identity node e_j we have $\tau \sim e_j$. Hence by 7.19 every element of Ψ found by 7.16 is active at an ancestor of some identity node. The result now follows from 7.20. □

7.23 Despite the power of Algorithm 7.16, its efficiency can be increased still further. Upon creating a node v of T_2, Algorithm 7.16 initialises a partition π_v as the discrete partition of $W(v)$. In this sense it assumes no prior knowledge of Ψ_v. However, if we have a set

$\{\gamma_1, \cdots, \gamma_m\}$ of previously discovered elements of Ψ, then some of them, say $\{\gamma_1, \cdots, \gamma_q\}$, may be in Ψ_ν. Then clearly we can initialise $\pi_k := (\theta_{\gamma_1} \vee \cdots \vee \theta_{\gamma_q})\big|_{W(\nu)}$ without losing identity nodes. In fact, we could set $\pi_k := \theta_\Lambda\big|_{W(\nu)}$ where $\Lambda = \langle \gamma_1, \cdots, \gamma_m \rangle_\nu$, but in practice this seems to be rarely worth the additional computation required.

Several points are worth mentioning here.

(1) Only the partitions $\pi^{(i)} = \theta_{\gamma_i}$ need to be stored. In fact, only the non-trivial cells of $\pi^{(i)}$ are required.

(2) There is no need to store all the elements of Ψ discovered. Storing too many elements can actually slow down the algorithm since the constant initialisation of partitions π_k may become too laborious. In practice, we can choose a small integer J and store only the first J elements of Ψ discovered.

These ideas give rise to the following algorithm, which is a variation on 7.16.

7.24 Algorithm: Find the identity nodes of T.

(1) Set k := 0; s := 0; t := 0.

(2) Compute $Z := W([v_1, \cdots, v_k])$. If $Z = \phi$ go to step (10).

(3) Set π_k := discrete partition of Z.
For $1 \le i \le t$ such that $\pi^{(i)}$ fixes $[v_1, \cdots, v_k]$ set $\pi_k := \pi_k \bar{\vee} \pi^{(i)}$.

(4) Set C := first cell of π_k not yet chosen;
v_{k+1} := smallest point in C;
k := k + 1.

(5) If $k < n$ go to step (2).

(6) We have found a terminal node $\tau = [v_1, \cdots, v_n]$.
If $\tau \sim e_j$ for some j $(1 \leq j \leq s)$ go to step (8).

(7) Set $s := s + 1$; $e_s := \tau$.
Go to step (10).

(8) Compute γ such that $\tau = e_j^{\gamma}$. Set $k := |\tau - e_j|$.
For $0 \leq i < k$ set $\pi_i := \pi_i \bar{v} \theta_\gamma$.

(9) If $t = J$ go to step (10).
Otherwise set $t := t + 1$; $\pi^{(t)} := \theta_\gamma$.

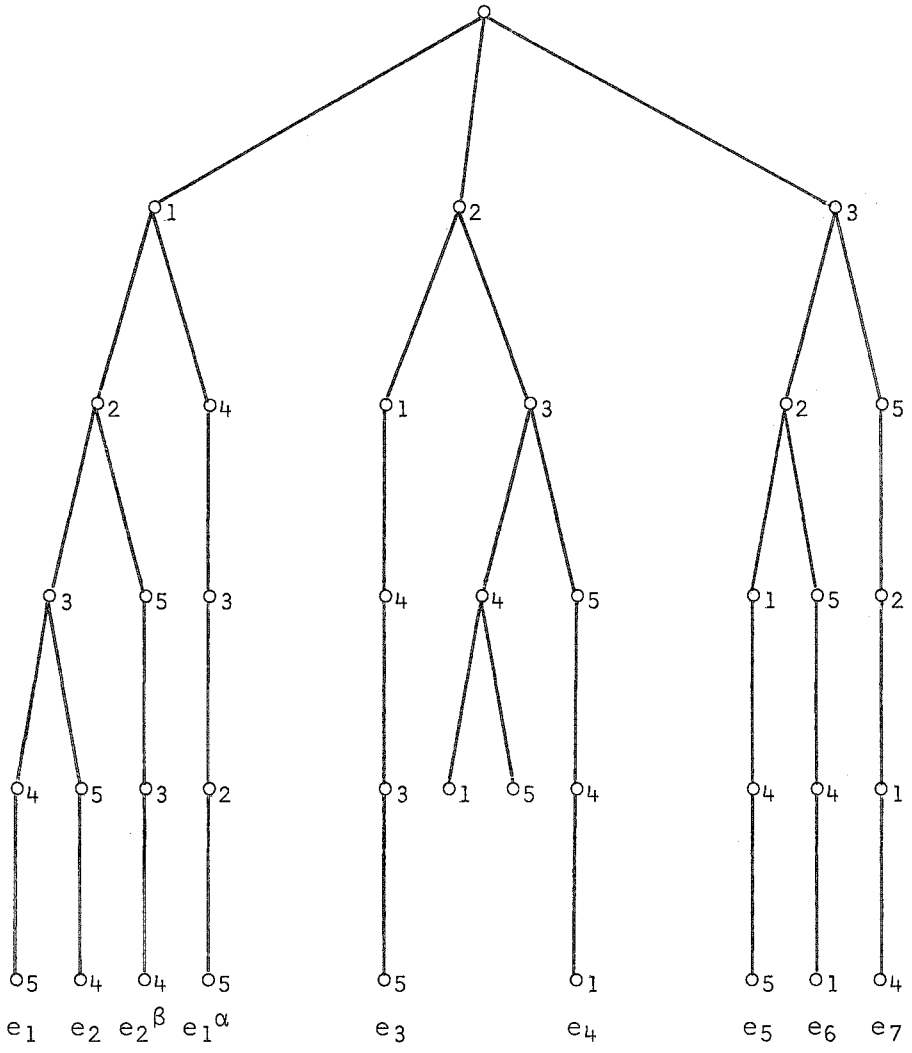
(10) If $k = 0$ stop.
Set $k := k - 1$.

(11) If all cells of π_k have been chosen go to step (10); otherwise go to step (4).

If $J = 0$, then Algorithm 7.24 is identical to Algorithm 7.16. If $J \geq 2$, then applying Algorithm 7.24 to the example of 6.6 produces the program tree of Figure 7.3. In the process of the algorithm, we have only needed to store the partitions for (3 5) and (2 4).

The *activity* of an element of Ψ discovered by 7.24 is defined as for 7.16.

7.25 Theorem: *Let* Y *be the set of elements of* Ψ *discovered by Algorithm 7.24 (for any* J*) which are active at ancestors of the first terminal node* e_1. *Then* Y *generates* Ψ *and* $|Y| \leq n - p$, *where* Ψ *has* p *orbits.*



$\alpha = (2\ 4)$, $\beta = (3\ 5)$

Figure 7.3

Proof: When the ancestors of e_1 are created during 7.24 we have $t = 0$ since no elements of Ψ have been found. The proof of 7.20 can therefore be applied. □

Let T_3 denote the program tree produced by Algorithm 7.24. If J is large enough, the number of terminal nodes of T_3 seems to be typically of order $r + n$. However, no bound better than that for T_2 has been proven. For program trees with a lot of endnodes which are not terminal nodes, T_3 is often vastly smaller than T_2, since the size of subtrees without terminal nodes can be reduced.

7.26 We now turn to a variation on Algorithms 7.3, 7.16 and 7.24. In all of these algorithms, it is necessary to store the full set of identity nodes. If this set is required exactly, there seems to be no alternative, since otherwise further identity nodes could not be positively identified. However, in some applications a larger set of terminal nodes, known to contain the identity nodes, will be sufficient. In these cases we can store a subset of the identity nodes. Terminal nodes which are equivalent to identity nodes that are not stored will then be recognised as "possibly an identity node".

One method which appears to work very well is to choose an integer $L \geq 0$ and to store the first identity node e_1 and the latest L terminal nodes which are "possibly identity nodes". The reason for storing e_1 is that then Theorems 7.9, 7.20 and 7.25 will still hold for this identity node.

7.27 The simplest case here is when $L = 0$ so that only the first identity node is stored. When Algorithm 7.24 with this change is applied to the example of 6.6, the program tree T_4 of Figure 7.4 is

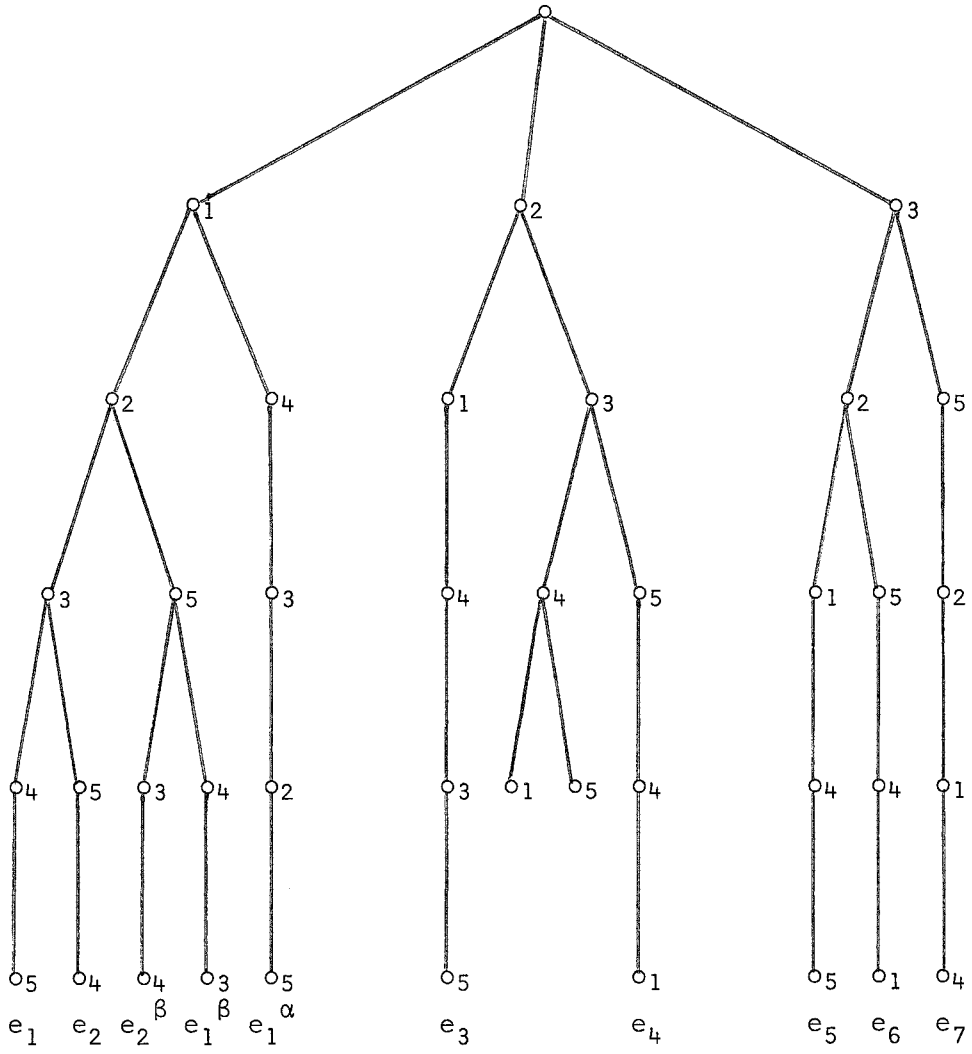

1 2 3
2 4 1 3 2 5
3 5 3 4 5 1 5 2
4 5 3 4 2 3 4 5 4 1 5 4 4 1
5 4 4 3 5 5 1 5 1 4
e1 e2 e2β e1β e1α e3 e4 e5 e6 e7
α = (2 4), β = (3 5)

Figure 7.4

produced. It is seen (in this example at least) that T_4 is only marginally larger than T_3. An advantage of this case (L = 0) is that the set Y of Theorem 7.25 contains all the elements of Ψ discovered by 7.24. In fact, it is the set Y' which would be produced by Algorithm 4.9 from the set Ω_0 of all elements of Ψ discovered by Algorithm 7.3 with L = 0.

7.28 After work on this chapter was completed, it was discovered that a method akin to that of Algorithm 7.16 had previously been used in a special case by Arlazarov et al. [2], who were concerned with the problem of canonically labelling a graph. However, to the best of our knowledge, Algorithm 7.24 and all our results on the generation of Ψ and on the size of T_1 and T_2 are original.

7.29 In practical problems it is very common for many nodes of the program tree T to have only one successor. In other words, for many nodes $\nu \in T$, we have $|W(\nu)| = 1$. For such nodes there is clearly no need to have a set U_k (as in 6.5 or 7.3) or a partition π_ν (as in 7.16 or 7.24) since these will always be trivial. Similarly, on "backtracking" out of the subtree $T(\nu)$ there is no need to examine ν since there cannot be further paths downwards from ν. Therefore we can consider such nodes (excepting the root) to be omitted from the tree. For example, the tree T of Figure 7.5 can be reduced to the tree $\tilde{T}$ of Figure 7.6.

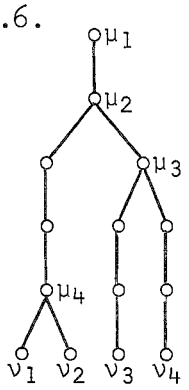


Figure 7.5

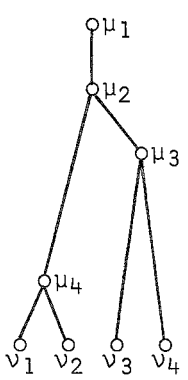


Figure 7.6

Since T is determined by its endnodes (all other nodes are ancestors of endnodes), it is trivial to reconstruct T from $\tilde{T}$ and so both trees contain the same information. "Reduced" trees like $\tilde{T}$ can be analysed by generalising the ideas of defining and successor functions. For example, the tree $\tilde{T}$ of Figure 7.6 is described by a generalised successor function $\tilde{F}$ such that

$$F(\mu_1) = \{\mu_2\}$$
$$F(\mu_2) = \{\mu_3, \mu_4\}$$
$$F(\mu_3) = \{\nu_3, \nu_4\}$$
$$F(\mu_4) = \{\nu_1, \nu_2\}$$
$$F(\nu_i) = \phi \ (i = 1, 2, 3, 4).$$

7.30 There is no reason why we could not delete just some of the nodes of T with one successor. If this is done so that $\tilde{T}$ is still invariant under Ψ, all the results of Chapters Six and Seven can be simply adapted to this case. Such reduced trees will occur in our applications in later chapters.

CHAPTER EIGHT

GRAPH ISOMORPHISM PROBLEMS

8.1 There are several related problems which fall under the general title of "graph isomorphism problems". The main ones can be stated approximately as follows.

Let G_1, G_2 and G be labelled graphs.

P1. (a) Are G_1 and G_2 isomorphic?

(b) If G_1 and G_2 are isomorphic, find one (or all) isomorphisms between them.

P2. Find a canonical labelling of G.

P3. Determine the group $\Gamma(G)$.

P4. Find one (or all) subgraphs of G_1 isomorphic to G_2.

P5. Find the common subgraphs (or maximal common subgraphs) of G_1 and G_2.

8.2 Apart from their obvious impact on graph-theoretic research, solutions to these problems have many direct practical applications. A much-quoted example concerns the storage and recognition of chemical compound structures [42, 43, 52], where a molecule can be represented as a graph with points and edges labelled by atom type and bond type respectively. Another application is in pattern recognition [74], where shapes can often be described in terms of graphs and need to be recognised despite their orientation and distortion.

8.3 Problems P4 and P5 will not be considered in this thesis,

although future research may be directed towards an extension of our procedures to these cases. Problem P4, usually called the "subgraph isomorphism problem" has received attention from Sussenguth [70], Penny [52], Levi [39], Levi and Luccio [41], Berztiss [5] and Ullmann [75]. The special case where G_1 and G_2 are trees has been considered by Matula [45]. Problem P5 has been treated only rarely, for example by Levi [39].

8.4 Proposed methods for solving problem P1 generally fall into one of two broad classes. The first approach, which we shall call *approach A*, treats G_1 and G_2 together. In the usual system, G_1 is relabelled in some way and then an attempt is made to relabel G_2 in such a way that G_1 and G_2 become identical.

The second approach, *approach B*, is to devise a map f from $\underset{\sim}{G}(V)$ into some convenient set Ω such that $f(G_1) = f(G_2)$ if and only if G_1 and G_2 are isomorphic. Unsuccessful or conjectural suggestions for f(G) in the past have included the characteristic polynomial of the adjacency matrix of G [10, 21, 59] and certain more general matrix functions [44, 73]. More success has been had in devising maps f as follows.

8.5 Let $f : \underset{\sim}{G}(V) \to \underset{\sim}{G}(V)$ be a map such that for each $G \in \underset{\sim}{G}(V)$ and $\gamma \in S_n$ we have

(1) f(G) is isomorphic to G, and

(2) $f(G^\gamma) = f(G)$.

f(G) can be called the *canonical labelling* of G. Its computation is the subject of problem P2.

8.6 The practical choice between these two basic approaches will depend on the application required. If graphs are to be compared in pairs only, then, under the current state of the art, approach A will undoubtedly be the more efficient. However, if larger collections of graphs need to be compared this will not necessarily be so.

Suppose we have a collection of N graphs which we wish to divide into isomorphic families. If the number of such families is almost as large as N and each comparison of two (labelled or unlabelled) graphs gives only a yes/no answer, approximately $\binom{N}{2}$ such comparisons are required. Define average execution times as follows.

t_1 : for comparing two unlabelled graphs

t_2 : for comparing two labelled graphs

t_3 : for canonically labelling a graph.

Approaches A and B will then take approximate times t_A and t_B, where

$$t_A = t_1\binom{N}{2} \text{ and}$$
$$t_B = Nt_3 + t_2\binom{N}{2}.$$

Hence, as $N \to \infty$, $t_B/t_A \to t_2/t_1$ which, for existing algorithms, is considerably less than one.

8.7 The great majority of existing algorithms for solving problem P1, whether by approach A or approach B, can be described in terms of a *canonical map*. This is defined to be a map

$$g : \underset{\sim}{G}(V) \to 2^{\underset{\sim}{G}(V)} \setminus \{\phi\}$$

such that for $G \in \underset{\sim}{G}(V)$ and $\gamma \in S_n$ we have

(1) $g(G^\gamma) = g(G)$ and

(2) G is isomorphic to every member of $g(G)$.

8.8 In terms of a canonical map g, approach A to solving problem P1 can be described as follows.

(1) Find one member G_1' of $g(G_1)$.

(2) Search $g(G_2)$ in some systematic fashion for a labelled graph identical to G_1'.

Commonly, steps (1) and (2) are carried out together, and intermediate information is used to help the search in (2). However, since we will not be particularly concerned with approach A, we will not go into these details here.

8.9 A canonical map g can also be used to canonically label a graph G. Firstly, we must devise a total order on $\underset{\sim}{G}(V)$. For example, we can apply the usual ordering of the integers by writing an adjacency matrix row-by-row as an n^2-bit binary number. Another simple method uses the incidence matrix [49, 55] in a similar way.

Relative to whatever order on $\underset{\sim}{G}(V)$ we have chosen, we can define a canonical labelling of $G \in \underset{\sim}{G}(V)$ by

$$f(G) = \max g(G).$$

The first use of this method was probably by Nagle [48], who defined g(G) to be the set of all labelled graphs isomorphic to G. A better choice was made by Heap [24], who required each member of g(G) to have its points in ascending order of degree. A similar

system, counting triangles as well as edges, was used by Baker et al. [3] when generating 9-point graphs.

8.10 Clearly the efficiency of any of these techniques will depend heavily on the choice of the canonical map g. A great many such maps have been used, explicitly or not, in published algorithms. However, almost all of them fall in the class we now describe.

8.11 Let $\mathcal{W}: \underset{\sim}{G}(V) \times Q(V) \to 2^V$ be a map such that the following hold for each $G \in \underset{\sim}{G}(V)$ and $\nu \in Q(V)$.

(1) $\mathcal{W}(G, \nu) \subseteq V \backslash \nu$.

(2) $\mathcal{W}$ is independent of the labelling of G (as defined in 6.26).

(3) The program tree T_G with defining function $\mathcal{W}(G, \cdot)$ has at least one terminal node.

Since every terminal node of T_G is in $Q^{(n)}(V)$, it corresponds to an ordering of V and hence to a relabelling of G. If we define g(G) to be the set of labelled graphs corresponding to the terminal nodes of T_G then g is canonical by 6.27.

Explicit uses of this method for finding a canonical map have been given by Berztiss [5], Proskurowski [49, 55], Ullmann [75] and Arlazarov et al. [2]. However, most of the so-called "partitioning" procedures also fall into this class, as we shall demonstrate shortly.

8.12 Obviously, any terminal nodes of T_G which are equivalent under $\Gamma(G)$ correspond to the same labelled graph. Consequently at

most one member of each equivalence class under $\Gamma(G)$ is required for the determination of $g(G)$. Therefore, any of the methods described in Chapter Seven for reducing the size of T_G can be used. However, except as mentioned in 7.28, they have not been used in any published algorithm that we know of. This is the main reason why we believe our own algorithms (described in the next chapter) to be superior to any previous algorithms.

8.13 We now proceed to give a formal account of "partitioning" procedures and show how they lead to maps $\mathcal{W}$ of the type described in 8.11. We first require a few definitions.

8.14 Let $\pi = [C_1|C_2|\cdots|C_k] \in \Pi(V)$ and $\gamma \in S_n$. Then π^γ denotes the ordered partition $[C_1{}^\gamma|C_2{}^\gamma|\cdots|C_k{}^\gamma]$.

Let Δ be a totally ordered set. A map

$$\mathcal{F} : \underset{\sim}{G}(V) \times \tilde{\Pi}(V) \times V \to \Delta$$

will be called an *indicator function* if, for each $G \in \underset{\sim}{G}(V)$, $\pi \in \tilde{\Pi}(V)$, $v \in V$, $\gamma \in S_n$, we have

$$\mathcal{F}(G^\gamma, \pi^\gamma, v^\gamma) = \mathcal{F}(G, \pi, v).$$

Similarly a map

$$\mathcal{P} : \underset{\sim}{G}(V) \times \tilde{\Pi}(V) \to \tilde{\Pi}(V)$$

will be called a *partition function* if, for each $G \in \underset{\sim}{G}(V)$, $\pi \in \tilde{\Pi}(V)$, $\gamma \in S_n$, we have

$$\mathcal{P}(G^\gamma, \pi^\gamma) = (\mathcal{P}(G, \pi))^\gamma.$$

Let $\underset{\sim}{\mathcal{I}}(V)$ and $\underset{\sim}{\mathcal{P}}(V)$ denote, respectively, the families of all indicator functions and partition functions for V.

8.15 The sets $\underset{\sim}{\mathcal{I}}(V)$ and $\underset{\sim}{\mathcal{P}}(V)$ are closely related as follows.

Let $\mathcal{I} \in \underset{\sim}{\mathcal{I}}(V)$. Then we can find a corresponding partition function $\mathcal{P} = \mathcal{P}(\mathcal{I})$ where for $G \in \underset{\sim}{G}(V)$ and $\pi \in \tilde{\Pi}(V)$, $\mathcal{P}(G, \pi)$ is the ordered partition whose cells contain points with the same value of $\mathcal{I}(G, \pi, v)$ and are in the order induced from Δ.

Similarly, let $\mathcal{P} \in \underset{\sim}{\mathcal{P}}(V)$. Then we can find a corresponding indicator function $\mathcal{I} = \mathcal{I}(\mathcal{P})$ as follows. Let Δ be the natural numbers. For $G \in \underset{\sim}{G}(V)$, $\pi \in \tilde{\Pi}(V)$ and $v \in V$ let $\mathcal{I}(G, \pi, v) = i$, where v is in the i-th cell of $\mathcal{P}(G, \pi)$.

The following lemma is trivial.

8.16 Lemma: *Let* $\mathcal{I} \in \underset{\sim}{\mathcal{I}}(V)$, $\mathcal{P} \in \underset{\sim}{\mathcal{P}}(V)$. *Then* $\mathcal{I}(\mathcal{P}) \in \underset{\sim}{\mathcal{I}}(V)$ *and* $\mathcal{P}(\mathcal{I}) \in \underset{\sim}{\mathcal{P}}(V)$. *Furthermore,*

(1) $\mathcal{P}(\mathcal{I}(\mathcal{P})) = \mathcal{P}$.

(2) *For any* $G \in \underset{\sim}{G}(V)$, $\pi \in \tilde{\Pi}(V)$ *and* $v_1, v_2 \in V$,

$$\mathcal{I}(\mathcal{P}(\mathcal{I}))(G, \pi, v_1) \leq \mathcal{I}(\mathcal{P}(\mathcal{I}))(G, \pi, v_2)$$

iff

$$\mathcal{I}(G, \pi, v_1) \leq \mathcal{I}(G, \pi, v_2). \qquad \square$$

8.17 We have already mentioned (8.9) the indicator function used by Heap [24]. In this case we have

$$\mathcal{I}(G, \pi, v) = d_G(v).$$

Similar functions used by other authors include

(1) the number of points at a given distance from v [40,76],

(2) the number of points adjacent to v (or at a given distance from v) which lie in a given cell of π[40, 76], and

(3) the components corresponding to v in the eigenvectors of the adjacency matrix of G [36].

The next few results show how partition functions can be combined to give other partition functions.

8.18 Theorem: *Let* $\mathcal{P}_1, \mathcal{P}_2 \in \underset{\sim}{\mathcal{P}}(V)$. *Then* $\mathcal{P}_2(\mathcal{P}_1) \in \underset{\sim}{\mathcal{P}}(V)$ *where* $(\mathcal{P}_2(\mathcal{P}_1))(G, \pi) = \mathcal{P}_2(G, \mathcal{P}_1(G, \pi))$, *for* $G \in \underset{\sim}{G}(V)$ *and* $\pi \in \tilde{\Pi}(V)$.

Proof: For $\gamma \in S_n$,

$$\mathcal{P}_2(G^\gamma, \mathcal{P}_1(G^\gamma, \pi^\gamma)) = \mathcal{P}_2(G^\gamma, (\mathcal{P}_1(G, \pi))^\gamma)$$

$$= (\mathcal{P}_2(G, (\mathcal{P}_1(G, \pi)))^\gamma. \qquad \square$$

For ordered partitions $\pi_1, \pi_2 \in \tilde{\Pi}(V)$ we define $\pi_1 \wedge \pi_2$ to be the meet of the unordered partitions corresponding to π_1 and π_2, with the cells in the order induced from π_1 and π_2. Precisely, if

$$(\pi_1 \wedge \pi_2)(i) = \pi_1(i_1) \cap \pi_2(i_2)$$

and

$$(\pi_1 \wedge \pi_2)(j) = \pi_1(j_1) \cap \pi_2(j_2),$$

then $i < j$ iff either $i_1 < j_1$ or $i_1 = j_1$ and $i_2 < j_2$.

8.19 Lemma: *For any* $\gamma \in S_n$ *and* $\pi_1, \pi_2 \in \widetilde{\Pi}(V)$, $\pi_1^{\gamma} \wedge \pi_2^{\gamma} = (\pi_1 \wedge \pi_2)^{\gamma}$.

Proof: Trivial. □

8.20 Theorem: *Let* $\mathcal{P}_1, \mathcal{P}_2 \in \underset{\sim}{\mathcal{P}}(V)$. *Then* $\mathcal{P}_1 \wedge \mathcal{P}_2 \in \underset{\sim}{\mathcal{P}}(V)$ *where*

$$(\mathcal{P}_1 \wedge \mathcal{P}_2)(G, \pi) = \mathcal{P}_1(G, \pi) \wedge \mathcal{P}_2(G, \pi),$$

for $G \in \underset{\sim}{G}(V)$, $\pi \in \widetilde{\Pi}(V)$.

Proof: For any $\gamma \in S_n$,

$$\mathcal{P}_1(G^{\gamma}, \pi^{\gamma}) \wedge \mathcal{P}_2(G^{\gamma}, \pi^{\gamma}) = (\mathcal{P}_1(G, \pi))^{\gamma} \wedge (\mathcal{P}_2(G, \pi))^{\gamma}$$

$$= (\mathcal{P}_1(G, \pi) \wedge \mathcal{P}_2(G, \pi))^{\gamma}$$

by 8.19. □

8.21 Theorem: *Let* $\gamma \in \Gamma_{\pi}$ *where* $\Gamma = \Gamma(G)$, $G \in \underset{\sim}{G}(V)$, $\pi \in \widetilde{\Pi}(V)$, *let* $\mathcal{P} \in \underset{\sim}{\mathcal{P}}(V)$.

Then $[\mathcal{P}(G, \pi)]^{\gamma} = \mathcal{P}(G, \pi)$.

Proof: If $\gamma \in \Gamma_{\pi}$, then $G^{\gamma} = G$ and $\pi^{\gamma} = \pi$. □

The most common method of obtaining partition functions is via indicator functions as shown in 8.15. From these partition functions others can be constructed using 8.18 or 8.20. The following few results indicate a related method which was first treated systematically by Tinhofer [72] but used previously by Unger [76] and other authors.

Let $\mathcal{R}_*(G, \pi)$ and $\mathcal{R}_K(G, \pi)$ denote, respectively, the

resulting partitions when Algorithms 5.27 and 5.32 are applied to $G \in \underset{\sim}{G}(V)$ and $\pi \in \tilde{\Pi}(V)$.

8.22 Theorem: *$\mathcal{R}_* \in \underset{\sim}{\mathcal{P}}(V)$ and $\mathcal{R}_K \in \underset{\sim}{\mathcal{P}}(V)$ for any $K > 0$.*

Proof: The vector $\underset{\sim}{d}(v, \pi)$ of 5.26 is clearly an indicator function and so $\mathcal{R}_* \in \underset{\sim}{\mathcal{P}}(V)$ by 8.18 and 8.20. Similarly, the transformation of $\tilde{\pi}$ from step (5) to step (7) of Algorithm 5.32 constitutes a partition function. □

8.23 Let $c : \underset{\sim}{G}(V) \to \underset{\sim}{G}(V)$ be a map such that for any $G \in \underset{\sim}{G}(V)$, $\gamma \in S_n$ we have

$$c(G^\gamma) = (c(G))^\gamma.$$

For example $v_1, v_2 \in V$ might be adjacent in $c(G)$ exactly when $\partial(v_1,v_2) = k$ in G (for some fixed k). If $\gamma \in \Gamma(G)$, then $(c(G))^\gamma = c(G^\gamma) = c(G)$ and so $\gamma \in \Gamma(c(G))$. Hence $\Gamma(G) \leq \Gamma(c(G))$.

8.24 Theorem: *Let $c : \underset{\sim}{G}(V) \to \underset{\sim}{G}(V)$ be a map satisfying 8.23. Let $\mathcal{P} \in \underset{\sim}{\mathcal{P}}(V)$. Then $\mathcal{P}_c \in \mathcal{P}(V)$ where for $G \in \underset{\sim}{G}(V)$ and $\pi \in \tilde{\Pi}(V)$,*

$$\mathcal{P}_c(G, \pi) = \mathcal{P}(c(G), \pi).$$

Proof: For any $\gamma \in S_n$,

$$\mathcal{P}(c(G^\gamma), \pi^\gamma) = \mathcal{P}((c(G))^\gamma, \pi^\gamma)$$

$$= (\mathcal{P}(c(G), \pi))^\gamma.$$ □

8.25 Theorem: [72] *Let $G \in \underset{\sim}{G}(V)$ and let π_0 be the unit partition of V. Then there is a sequence $c_1, c_2, \cdots, c_k$ of maps satisfying 8.23 such that the cells of π_k are orbits of $\Gamma(G)$, where*

$\pi_i = \mathcal{R}_*(c_i(G), \pi_{i-1})$ for $1 \le i \le k$. □

We now demonstrate how partition functions can help us to find functions $\mathcal{W}$ satisfying 8.11.

An ordered partition $\pi \in \tilde{\Pi}(V)$ will be said to *fix* a sequence $\nu \in Q(V)$ if each element of ν is in a trivial cell of π.

8.26 Let $\mathcal{B} : \underset{\sim}{G}(V) \times Q(V) \to \tilde{\Pi}(V)$ be a map such that for $G \in \underset{\sim}{G}(V)$, $\nu \in Q(V)$, $\gamma \in S_n$ we have

(1) $\mathcal{B}(G^\gamma, \nu^\gamma) = (\mathcal{B}(G, \nu))^\gamma$, and

(2) $\mathcal{B}(G, \nu)$ fixes ν.

Given $\mathcal{P} \in \underset{\sim}{\mathcal{P}}(V)$, one such map can be found as follows. If $\nu = [v_1, \cdots, v_k]$, let $\pi = [v_1|v_2|\cdots|v_k|V\backslash\nu]$. Then we can take $\mathcal{B}(G, \nu) = \pi \wedge \mathcal{P}(G, \pi)$. Other similar schemes are possible.

8.27 Let $\mathcal{C} : \tilde{\Pi}(V) \times Q(V) \to 2^V$ be a map such that for $\nu \in Q(V)$, $\gamma \in S_n$ and $\pi \in \tilde{\Pi}(V)$ which fixes ν we have the following.

(1) $\mathcal{C}(\pi^\gamma, \nu^\gamma) = (\mathcal{C}(\pi, \nu))^\gamma$.

(2) If $|\nu| = n$, then $\mathcal{C}(\pi, \nu) = \phi$.

(3) If $|\nu| < n$, then $\mathcal{C}(\pi, \nu)$ is a cell of π not containing an element of ν.

For example, we might take $\mathcal{C}(\pi, \nu)$ to be the first cell of π not containing an element of ν, or the first such cell of smallest size.

8.28 Theorem: *Let the maps* $\mathcal{B}$ *and* $\mathcal{C}$ *satisfy 8.26 and 8.27 respectively. Then the map*

$$\mathcal{W} : \underset{\sim}{G}(V) \times Q(V) \to 2^V$$

defined by

$$\mathcal{W}(G, \nu) = \mathcal{C}(\mathcal{B}(G, \nu), \nu)$$

for $G \in \underset{\sim}{G}(V)$ *and* $\nu \in Q(V)$ *satisfies the conditions of 8.11.*

Proof: Condition (1) follows from 8.27 (3). If $\gamma \in S_n$, $G \in \underset{\sim}{G}(V)$, and $\nu \in Q(V)$, we have

$$\begin{aligned}
\mathcal{W}(G^\gamma, \nu^\gamma) &= \mathcal{C}(\mathcal{B}(G^\gamma, \nu^\gamma), \nu^\gamma) \\
&= \mathcal{C}((\mathcal{B}(G, \nu))^\gamma, \nu^\gamma) \\
&= (\mathcal{C}(\mathcal{B}(G, \nu), \nu))^\gamma \\
&= (\mathcal{W}(G, \nu))^\gamma,
\end{aligned}$$

so that $\mathcal{W}$ satisfies condition (2). Finally, the program tree T_G contains terminal nodes since $\mathcal{W}(G, \nu) \neq \phi$ if $|\nu| < n$. □

8.29 Given the map $\mathcal{W}$ defined in 8.28 we can define a function f satisfying 8.5 as we indicated in 8.9. However, in practice, the following method may be more convenient. Suppose we have decided on a total ordering of $\underset{\sim}{G}(V)$.

For $G \in \underset{\sim}{G}(V)$ define T_G as in 8.11 (3) and let X(G) be the set of its terminal nodes. For any $\tau \in X(G)$ define G^τ to be the labelled graph formed by labelling the vertices of G in the order they appear in $\mathcal{B}(G, \tau)$. Then define

$$f(G) = \max\{G^\tau \mid \tau \in X(G)\}.$$

8.30 Theorem: *The function* $f : \underset{\sim}{G}(V) \to \underset{\sim}{G}(V)$ *defined above satisfies 8.5.*

Proof: For any $G \in \underset{\sim}{G}(V)$, $f(G)$ and G are obviously isomorphic. Now let $\gamma \in S_n$. Then if $\tau \in X(G)$, $\tau^\gamma \in X(G^\gamma)$ by 6.27.

But $\mathcal{B}(G^\gamma, \tau^\gamma) = (\mathcal{B}(G, \tau))^\gamma$ and so $(G^\gamma)^{\tau^\gamma} = G^\tau$.

Therefore $f(G^\gamma) = f(G)$. □

8.31 If $G \in \underset{\sim}{G}(V)$, $\tau \in X(G)$ and $\gamma \in \Gamma(G)$, then $G^{\tau^\gamma} = G^\tau$. Consequently any of the methods described in Chapter Seven can be used to eliminate terminal nodes equivalent under $\Gamma(G)$ without changing $f(G)$. These methods have an additional advantage in that a small set of generators for $\Gamma(G)$ can be found, for example, as described in 7.25.

8.32 Very commonly in implementing these ideas we find that $\mathcal{W}(G, \nu)$ consists of just one point for many nodes of the program tree. Nodes of this type can be removed from the tree, as described in 7.29. A very convenient arrangement for doing this is as follows. The function $\mathcal{C}$ of 8.27 can be defined so that $\mathcal{C}(\pi, \nu)$ will be a non-trivial cell of π if there are any. Furthermore, the map $\mathcal{B}$ of 8.26 can be defined so that if ν_1 is an ancestor of ν_2 in the tree T_G we have

$$\mathcal{B}(G, \nu_2) \leq \mathcal{B}(G, \nu_1),$$

and if $\mathcal{B}(G, \nu_1)$ is discrete,

$$\mathcal{B}(G, \nu_2) = \mathcal{B}(G, \nu_1).$$

In this situation the partition $\mathcal{B}(G, \tau)$ for $\tau \in X(G)$ can be found from the earliest ν of its ancestors for which $\mathcal{B}(G, \nu)$ is discrete. Later ancestors can be ignored.

CHAPTER NINE

A NEW CANONICAL LABELLING ALGORITHM

9.1 In this chapter we present several versions of a new algorithm for canonically labelling a graph and for determining its automorphism group. This algorithm was originally inspired by King's implementation [31] of the method of Parris and Read [50, 51], and retains a superficial similarity to this method. However, many improvements have been made. Most importantly, the methods of Chapter Seven have been applied, making the algorithm useful for graphs with large automorphism groups. Secondly, the use of Algorithm 5.32 instead of 5.27 has effected a great increase in efficiency. Finally, several ad hoc features to be described later have been incorporated. Once the algorithm has been presented and examples given, we treat the problem of efficiency in some detail. Evidence is presented in support of our claim that for large random graphs the algorithm is close to the fastest possible. All of this chapter is original.

9.2 The basic structure of the algorithm is as described in 8.29 and 8.32. Therefore, our first step will be to define maps $\mathcal{C}$, $\mathcal{B}$ and $\mathcal{W}$ satisfying 8.27, 8.26 and 8.11 respectively.

9.3 Define a map $\mathcal{C} : \tilde{\Pi}(V) \times Q(V) \to 2^V$ as follows. Let $\nu \in Q(V)$, $\pi \in \tilde{\Pi}(V)$.

(1) If π does not fix ν, or $|\nu| = n$, define $\mathcal{C}(\pi, \nu) = \phi$.

(2) If π fixes ν and π is not discrete, define $\mathcal{C}(\pi, \nu)$ to be the first of the non-trivial cells of π of smallest size.

(3) If π is discrete and $|\nu| < n$, define $\mathcal{C}(\pi, \nu)$ to be the

first cell of π not containing an element of ν.

9.4 Lemma: *$\mathcal{C}$ satisfies the conditions of 8.27.*

Proof: Trivial. □

9.5 Define a map $\mathfrak{D} : \tilde{\Pi}(V) \times V \to \tilde{\Pi}(V)$ as follows. Let $v \in V$, $\pi \in \tilde{\Pi}(V)$.

(1) If v is in a trivial cell of π, define $\mathfrak{D}(\pi, v) = \pi$.

(2) If $\pi = [C_1|\cdots|C_\ell]$ and v is in the non-trivial cell C_r, define

$$\mathfrak{D}(\pi, v) = [C_1|\cdots|C_r\backslash\{v\}|\cdots|C_\ell|\{v\}].$$

9.6 Lemma: *Let $v \in V$, $\pi \in \tilde{\Pi}(V)$ and $\gamma \in S_n$. Then*

$$\mathfrak{D}(\pi^\gamma, v^\gamma) = (\mathfrak{D}(\pi, v))^\gamma.$$

Proof: In case (1) the lemma is trivial. In case (2) we have

$$\mathfrak{D}(\pi^\gamma, v^\gamma) = [C_1{}^\gamma|\cdots|C_r{}^\gamma\backslash\{v^\gamma\}|\cdots|C_\ell{}^\gamma|\{v^\gamma\}]$$

$$= (\mathfrak{D}(\pi, v))^\gamma.$$ □

9.7 As before, let $\mathcal{R}_K(G, \pi)$ denote the result of Algorithm 5.32 when applied to $G \in \underset{\sim}{G}(V)$ and $\pi \in \tilde{\Pi}(V)$. Define a map $\mathfrak{B} : \underset{\sim}{G}(V) \times Q(V) \to \tilde{\Pi}(V)$ as follows. Let $G \in \underset{\sim}{G}(V)$ and $\nu \in Q(V)$.

(1) If $|\nu| = 0$, define $\mathfrak{B}(G, \nu) = \mathcal{R}_1(G, \pi_0)$, where π_0 is the unit partition of V.

(2) Suppose $\nu = [v_1, \cdots, v_k]$, where $0 < k \leq n$. Then define

$$\mathcal{B}(G, \nu) = \mathcal{R}_{\ell+1}(G, \mathcal{D}(\mathcal{B}(G, \mu), v_k)),$$

where

$$\mu = [v_1, \cdots, v_{k-1}] \quad \text{and} \quad \ell = |\mathcal{B}(G, \mu)|.$$

9.8 Lemma: $\mathcal{B}$ *satisfies the conditions of 8.26.*

Proof: If $|\nu| = 0$, the result follows trivially from 8.22. Otherwise it follows, by simple induction on $|\nu|$, from 8.22 and 9.6. □

9.9 Theorem: For any $G \in \underset{\sim}{G}(V)$ and $\nu \in Q(V)$ we have $\mathcal{B}(G, \nu) \simeq \pi$ where π is the coarsest element of $\Xi(G)$ which fixes ν.

Proof: If $|\nu| = 0$ the result follows from 5.34.

Suppose the theorem is true for $\mu = [v_1, \cdots, v_{k-1}]$ $(0 < k \leq n)$. Let $\nu = [v_1, \cdots, v_k] \in Q(V)$.

Then, by definition, $\mathcal{B}(G, \nu) = \mathcal{R}_{\ell+1}(G, \mathcal{D}(\pi_1, v_k))$ where $\pi_1 = \mathcal{B}(G, \mu)$ has ℓ cells. The induction hypothesis says that $\mathcal{B}(G, \mu) \simeq \pi_2$ where π_2 is the coarsest element of $\Xi(G)$ fixing μ.

Suppose $\mathcal{B}(G, \nu) \simeq \pi \in \Pi(V)$. By 5.36, π is the coarsest element of $\Xi(G)$ finer than $\mathcal{D}(\pi_1, v_k)$. But the coarsest element of $\Xi(G)$ fixing ν is finer than the coarsest fixing μ (trivially) and so is finer than $\mathcal{D}(\pi_1, v_k)$.

Hence π is the coarsest element of $\Xi(G)$ fixing ν. □

9.10 Following 8.28, define a map $\mathcal{W}: \underset{\sim}{G}(V) \times Q(V) \to 2^V$ by

$\mathcal{W}(G, \nu) = \mathcal{C}(\mathcal{B}(G, \nu), \nu)$ for any $G \in \underset{\sim}{G}(V)$ and $\nu \in Q(V)$. A canonical labelling $f(G)$ of G can then be defined as in 8.29. We have used a total ordering of $\underset{\sim}{G}(V)$ derived from a lexigraphic ordering of the adjacency matrices of its elements.

We shall find the following notation convenient. If $G \in \underset{\sim}{G}(V)$ and $\pi \in \tilde{\Pi}(V)$ is discrete, we define $G(\pi)$ to be the labelled graph formed by labelling the points of G in the order that they appear in π.

9.11 Clearly, any of the methods of Chapter Seven can be used to find the set $X(G)$ or a subset of $X(G)$ containing the identity nodes of T_G with respect to $\Gamma(G)$. The method which we will describe is based on 7.24 but altered so that $L = 0$, as described in 7.27.

For convenience, we list a few of the variables used in the description of the algorithm and note their usage. For $0 \leq k \leq n$ define $\nu_k = [v_1, \cdots, v_k]$.

Then $\xi_k = \mathcal{B}(G, \nu_k)$.

$\varepsilon = \mathcal{B}(G, e_1)$ where e_1 is the first terminal node.

$\rho = \mathcal{B}(G, \tau)$ where τ is the terminal node for which G^τ is greatest so far.

π_k is the (ordered) partition of $\mathcal{W}(G, \nu_k)$ as in 7.24.

$\pi^{(i)}$ is the orbits partition of the i-th element of $\Gamma(G)$ discovered. Only the non-trivial cells of $\pi^{(i)}$ need be stored.

$J \leq 0$ is the maximum number of partitions $\pi^{(i)}$ to be stored.

$$h = \begin{cases} 0 & \text{if no terminal nodes have been found.} \\ |e_1 - \tau| & \text{if } \tau \text{ is the next terminal node to be found, and } \tau \neq e_1. \end{cases}$$

We use the conventions of 7.12 and 7.15 throughout.

9.12 Algorithm: Canonically label $G \in \underset{\sim}{G}(V)$.

(1) Set $k := 0$; $t := 0$; $h := 0$.

(2) Compute $\xi_k := \mathcal{B}(G, \nu_k)$, where $\nu_k = [v_1, \cdots, v_k]$.
If ξ_k is discrete go to step (6).

(3) Set $Z := \mathcal{C}(\xi_k, \nu_k)$, where $\nu_k = [v_1, \cdots, v_k]$.

(4) Set $\pi_k :=$ discrete partition of Z with cells in numerical order.
For $1 \leq i \leq t$ such that $\pi^{(i)}$ fixes $[v_1, \cdots, v_k]$ set
$\pi_k := \pi_k \bar{\vee} \pi^{(i)}$.

(5) Set C := first cell of π_k not yet chosen;
v_{k+1} := smallest element of C;
$k := k + 1$.
Go to step (2).

(6) If $h \neq 0$ go to step (7).
Set $\rho := \varepsilon := \xi_k$;
$h := k$.
Go to step (10).

(7) If $G(\varepsilon) \neq G(\xi_k)$ go to step (9).
Compute γ such that $\xi_k = \varepsilon^\gamma$.
Set $k := h$.
For $0 \leq i < k$ set $\pi_i := \pi_i \bar{\vee} \theta_\gamma$.

(8) If $t = J$ go to step (10).

Set $t := t + 1$; $\pi^{(t)} := \theta_\gamma$.

Go to step (10).

(9) If $G(\xi_k) \leq G(\rho)$ go to step (10).

Set $\rho := \xi_k$.

(10) If $k = 0$ stop.

Set $h := \min(h, k)$; $k := k - 1$.

(11) If all cells of π_k have been chosen go to step (10).

Otherwise go to step (5).

9.13 As an example we label the graph G of Figure 9.1 with $J \geq 2$.

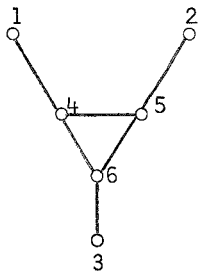


Figure 9.1

(1) $k = t = h = 0$.

(2) $\xi_0 = [1, 2, 3|4, 5, 6]$.

(3) $Z = \{1, 2, 3\}$.

(4) $\pi_0 = [1|2|3]$.

(5) $C = \{1\}$; $v_1 = 1$; $k = 1$.

(2) $\xi_1 = \mathcal{R}_3(G,[2, 3|4, 5, 6|1]) = [2, 3|5, 6|1|4]$.

(3) $Z = \{2, 3\}$.

(4) $\pi_1 = [2|3]$.

(5) $C = \{2\}$; $v_2 = 2$; $k = 2$.

(2) $\xi_2 = \mathcal{R}_5(G,[3|5, 6|1|4|2]) = [3|6|1|4|2|5].$

(6) $\rho = \varepsilon = [3|6|1|4|2|5]$

$h = 2.$

(10) $k = 1.$

(5) $C = \{3\}; v_2 = 3; k = 2.$

(2) $\xi_2 = \mathcal{R}_5(G,[2|5, 6|1|4|3]) = [2|5|1|4|3|6].$

(6) Go to (7).

(7) $G(\varepsilon) = G(\xi_2)$

$\gamma = (2\ 3)(5\ 6)$

$k = 2$

$\pi_0 = [1|2, 3]; \pi_1 = [2, 3].$

(8) $t = 1$

$\pi^{(1)} = \{2, 3|5, 6\}.$

(10) $k = 1.$

(11) Go to (10).

(10) $h = 1; k = 0.$

(11) Go to (5).

(5) $C = \{2, 3\}; v_1 = 2; k = 1.$

(2) $\xi_1 = \mathcal{R}_3(G,[1, 3|4, 5, 6|2]) = [1, 3|4, 6|2|5].$

(3) $Z = \{1, 3\}.$

(4) $\pi_1 = [1|3].$

(5) $C = \{1\}; v_2 = 1; k = 2.$

(2) $\xi_2 = \mathcal{R}_5(G,[3|4, 6|2|5|1]) = [3|6|2|5|1|4].$

(6) Go to (7).

(7) $G(\varepsilon) = G(\xi_2)$

$\gamma = (1\ 2)(4\ 5)$

$k = 1$

$\pi_0 = [1, 2, 3]$.

(8) $t = 2$

$\pi^{(2)} = \{1, 2|4, 5\}$.

(10) $k = 0$.

(11) Go to (10).

(10) *Stop:* $f(G) = G(\rho)$

where $\rho = [3|6|1|4|2|5]$.

9.14 The program tree of Algorithm 9.12 applied to the example of 9.13 is shown in Figure 9.2. For convenience, the nodes of the tree are labelled with the partitions ξ_k, with the cell $\mathcal{C}(\xi_k, \nu_k)$ underlined.

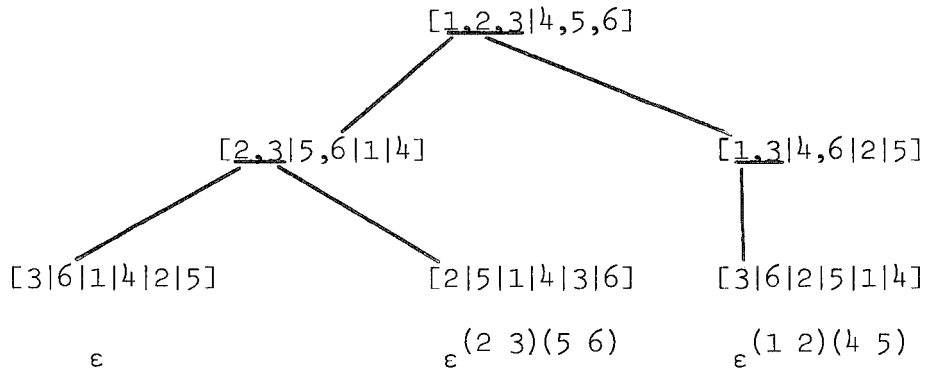


Figure 9.2

A more complicated example is shown in Figure 9.3, where $J \geq 3$.

9.15 Algorithm 9.12 provides a particularly convenient means

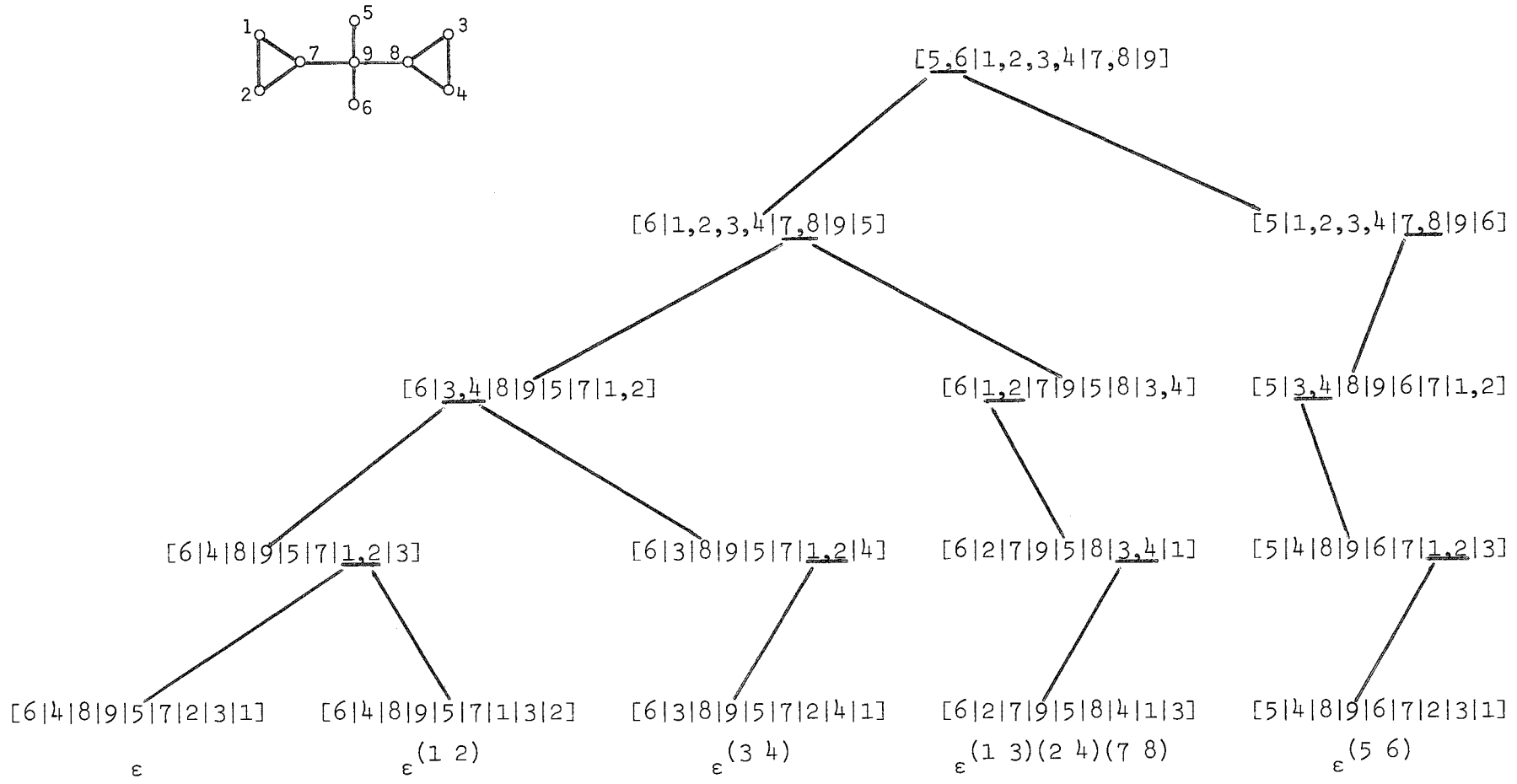

1
2
7
5
9
8
6
3
4
[5,6|1,2,3,4|7,8|9]
[6|1,2,3,4|7,8|9|5]
[5|1,2,3,4|7,8|9|6]
[6|3,4|8|9|5|7|1,2]
[6|1,2|7|9|5|8|3,4]
[5|3,4|8|9|6|7|1,2]
[6|4|8|9|5|7|1,2|3]
[6|3|8|9|5|7|1,2|4]
[6|2|7|9|5|8|3,4|1]
[5|4|8|9|6|7|1,2|3]
[6|4|8|9|5|7|2|3|1]
[6|4|8|9|5|7|1|3|2]
[6|3|8|9|5|7|2|4|1]
[6|2|7|9|5|8|4|1|3]
[5|4|8|9|6|7|2|3|1]
ε
ε(1 2)
ε(3 4)
ε(1 3)(2 4)(7 8)
ε(5 6)

Figure 9.3

of computing the automorphism group $\Gamma(G)$. While many published algorithms for graph isomorphism, for example by Levi [40] or Yang [80], can be used to find $\Gamma(G)$, there seem to be no algorithms other than our own for finding a small set of generators for $\Gamma(G)$. All other methods find each element of $\Gamma(G)$ individually, and so are practically useless if $|\Gamma(G)|$ is very large.

9.16 In Algorithm 9.12, suppose the first terminal node is $e_1 = [v_1, \cdots, v_n]$. For $0 \le j \le n$, define $\nu_j = [v_1, \cdots, v_j]$ and let ν_ℓ be the shortest such node for which $\mathcal{B}(G, \nu_\ell)$ is the discrete partition ε.

Suppose $0 \le j \le \ell$. Define $\Gamma^{(j)} = \Gamma_{\nu_j}$ where $\Gamma = \Gamma(G)$. If $j = 0$, consider the point of time when the algorithm terminates. Otherwise consider the instant when h is set to $j - 1$ for the first time at step (10). In other words, consider the algorithm immediately it has finished with ν_j and its descendants. At this point of time, define $\tilde{\pi}_j = \pi_j$ and let Y_j be the set of all elements of $\Gamma(G)$ so far discovered. Then from 7.20, 7.21 and 7.25 we have the following result.

9.17 Theorem: (a) Y_j *generates* $\Gamma^{(j)}$.

(b) $|Y_j| \le n - p_j$ *where* $\Gamma^{(j)}$ *has* p_j *orbits.*

(c) *The cells of* $\tilde{\pi}_j$ *are orbits of* $\Gamma^{(j)}$, $(j < \ell)$.

(d) $|\Gamma^{(j)}| = |\Gamma^{(j+1)}|\,|\tilde{\pi}_j(1)|$, $(j < \ell)$. □

We now consider a few simple means by which Algorithm 9.12 can be improved.

9.18 Let $0 < j \leq \ell$ and again consider the point of time when 9.12 has finished with ν_j and its descendants. For $0 \leq i < j$ the cells of the partition π_i at this stage are orbits of $\Gamma^{(j)}$. In the algorithm these partitions have been produced by applying the operation $\pi_i := \pi_i \bar{\vee} \theta_\gamma$ for each $\gamma \in Y_j$. We introduce a new partition $\xi \in \tilde{\Pi}(V)$ which is initially the discrete partition in numerical order. Each time we find an element $\gamma \in \Gamma(G)$ we set $\xi := \xi \bar{\vee} \theta_\gamma$. Then (at the point of time to which we are referring) the cells of ξ are the orbits of $\Gamma^{(j)}$ and so we can set π_{j-1} to the partition of $\mathcal{C}(\xi_{j-1}, \nu_{j-1})$ induced from ξ. This method has the added advantage that at the end of the algorithm the cells of ξ are the orbits of Γ.

9.19 Another source of inefficiency occurs at step (4) of 9.12. The computing of $\pi_k := \pi_k \bar{\vee} \pi^{(i)}$ for possibly many partitions $\pi^{(i)}$ will be unnecessary if no cell of π_k other than the first is ever chosen. This will be the case, for example, if the terminal node $\tau \neq e_1$ descended from the current node is absorbed onto an ancestor of e_1. Hence we can defer these computations until they are actually required.

9.20 Let ν be a node of the program tree produced by Algorithm 9.12 and let $\pi = \mathcal{B}(G, \nu)$. If $n - |\pi| \leq 5$, then by 5.19 and 9.9, $\mathcal{C}(\pi, \nu)$ is an orbit of Γ_π, where $\Gamma = \Gamma(G)$. Consequently all the terminal nodes descended from ν are equivalent. If e_1 is descended from ν, then we know that $G(\varepsilon) = G(\xi_k)$ at step (7) of 9.12 without computing $G(\xi_k)$, where ξ_k corresponds to a node descended from ν. On the other hand, if the terminal nodes descended from ν are not equivalent to e_1, they can be identified as such by examination of the first of them. In the following algorithm this change has been

handled by the variable q in a way best seen by examining the algorithm. It has led to more than a two-fold improvement in efficiency in many cases.

9.21 Algorithm: Canonically label $G \in \underset{\sim}{G}(V)$ and find generators for $\Gamma(G)$.

(1) Set k := 0; t := 0; h := 0; q := 0; m := 1;

ξ := discrete partition of V in numerical order.

(2) Compute $\xi_k := \mathcal{B}(G, \nu_k)$ where $\nu_k = [v_1, \cdots, v_k]$.

If ξ_k is discrete go to step (5).

If $n - |\xi_k| > 5$ set q := k + 1.

(3) Set $Z := \mathcal{C}(\xi_k, \nu_k)$ where $\nu_k = [v_1, \cdots, v_k]$;

π_k := discrete partition of Z in numerical order.

(4) Set C := first cell of π_k not yet chosen;

v_{k+1} := first element of C;

k := k + 1.

Go to step (2).

(5) If h > q go to step (8).

Compute $G(\xi_k)$.

If $h \neq 0$ go to step (7).

(6) Set $\rho := \varepsilon := \xi_k$;

h := k;

k := k - 1.

Go to step (11).

(7) If $G(\varepsilon) \neq G(\xi_k)$ go to step (10).

(8) Compute γ such that $\xi_k = \varepsilon^\gamma$.

Set $\xi := \xi \bar{v} \theta_\gamma$;

$k := h - 1$;

$\pi_k := \pi_k \bar{v} \theta_\gamma$.

(9) If $t = J$ go to step (11).

Set $t := t + 1$;

$\pi^{(t)} := \theta_\gamma$.

Go to step (11).

(10) If $G(\xi_k) > G(\rho)$ set $\rho := \xi_k$.

Set $k := q - 1$.

(11) If $k < 0$ *stop*: $f(G) = G(\rho)$.

If $k = h - 1$ or v_k is not in the first cell of π_k, go to step (13).

(12) For $1 \leq i \leq t$ such that $\pi^{(i)}$ fixes $[v_1, \cdots, v_k]$,

set $\pi_k := \pi_k \bar{v} \pi^{(i)}$.

(13) If $k < q$ set $q := k + 1$.

If not all the cells of π_k have been chosen go to step (4).

(14) Set $k := k - 1$.

If $k \geq h - 1$ go to step (11).

Set $h := k + 1$;

$m := m \times |\pi_h(1)|$.

(15) If $k < 0$ *stop*: $f(G) = G(\rho)$.

Set $\pi_k :=$ partition of $\mathcal{C}(\xi_k, \nu_k)$ induced from ξ.

If $k < q$ set $q := k + 1$.

Go to step (4).

9.22 Algorithm 9.21 produces the same program tree as does Algorithm 9.12 except for the occasional lopping of an unwanted subtree, as described in 9.20. Theorem 9.17 will still hold. In addition, at the termination of the algorithm we have $m = |\Gamma(G)|$ and the partition ξ gives the orbits of $\Gamma(G)$.

9.23 In many applications we may be interested in $\Gamma(G)$ but not in the canonical labelling $f(G)$. Clearly, in this case any terminal nodes of the program tree other than those equivalent to e_1 can be ignored. A convenient way in which many such nodes can be eliminated is by defining a function

$$\mathcal{L} : \underset{\sim}{G}(V) \times Q(V) \to \Delta$$

where Δ is any convenient set, and such that for $G \in \underset{\sim}{G}(V)$, $\nu \in Q(V)$ and $\gamma \in S_n$ we have

$$\mathcal{L}(G^\gamma, \nu^\gamma) = \mathcal{L}(G, \nu).$$

Now if $e_1 = [v_1, \cdots, v_n]$ and for some $\nu = [w_1, \cdots, w_k]$ we have

$$\mathcal{L}(G, \nu) \neq \mathcal{L}(G, [v_1, \cdots, v_k]),$$

then none of the terminal nodes descended from ν are equivalent to e_1. Hence the subtree $T(\nu)$ can be ignored.

A possible choice of $\mathcal{L}(G, \nu)$ is the quotient matrix of G induced by $\mathcal{B}(G, \nu)$, as defined in 5.23. This matrix has been used for related purposes by Levi [40], and Corneil and Gotlieb [11, 14]. However, because of the large amount of time needed to compute this matrix for each node of the tree, and because of the large amount of storage space required to hold as many as n of these matrices, we have adopted a simpler system.

Let $G \in \underset{\sim}{G}(V)$, $\nu \in Q(V)$, $\pi = \mathcal{B}(G, \nu)$, and define

$r_1 = |\pi|$,

$r_2 = i$, where $\mathcal{b}(\pi, \nu) = \pi(i)$,

$r_3 = |\pi(i)|$, and

r_4 = a computer word whose one-bits indicate the position of the trivial cells in π (see 3.10).

These four variables were chosen as being already available to the program. If π is discrete we define $\mathcal{L}(G, \nu) = 0$. Otherwise $\mathcal{L}(G, \nu)$ is a single machine word formed from $[r_1, r_2, r_3, r_4]$ using the shift and exclusive-or operations of the machine. Despite the simplicity of this system, it seems to be only rarely less powerful than the use of the quotient matrix.

9.24 Algorithm: Find generators for $\Gamma(G)$.

(1) Set k := 0; t := 0; h := 0; q := 0; m := 1;
ξ := discrete partition of V in numerical order.

(2) Compute $\xi_k := \mathcal{B}(G, \nu_k)$ where $\nu_k = [v_1, \cdots, v_k]$.
If ξ_k is discrete go to step (6).
Set $q_0 := q$.
If $n - |\xi_k| > 5$ set q := k + 1.
If h = 0 go to step (3).
If $\mathcal{L}(G, \nu_k) = \lambda_k$ go to step (4).
Set $q := q_0$.
Go to step (12).

(3) Set $\lambda_k := \mathcal{L}(G, \nu_k)$ where $\nu_k = [v_1, \cdots, v_k]$.

(4) Set $Z := \mathcal{C}(\xi_k, \nu_k)$ where $\nu_k = [v_1, \cdots, v_k]$;

$\pi_k :=$ discrete partition of Z in numerical order.

(5) Set C := first cell of π_k not yet chosen;

v_{k+1} := first element of C;

k := k + 1.

Go to step (2).

(6) If h = 0 go to step (7).

If $\lambda_k \neq 0$ go to step (12).

(7) If h > q go to step (10).

Compute $G(\xi_k)$.

If $h \neq 0$ go to step (9).

(8) Set $\varepsilon := \xi_k$;

h := k;

k := k - 1.

Go to step (13).

(9) If $G(\varepsilon) \neq G(\xi_k)$ go to step (12).

(10) Compute γ such that $\xi_k = \varepsilon^{\gamma}$.

Output γ.

Set $\xi := \xi \bar{v} \theta_\gamma$;

k := h - 1;

$\pi_k := \pi_k \bar{v} \theta_\gamma$.

(11) If t = J go to step (13).

Set $t := t + 1$; $\pi^{(t)} := \theta_\gamma$.

Go to step (13).

(12) Set k := q - 1.

(13) If $k < 0$ *stop*.

If $k = h - 1$, or v_k is not in the first cell of π_k, go to step (15).

(14) For $1 \leq i \leq t$ such that $\pi^{(i)}$ fixes $[v_1, \cdots, v_k]$ set $\pi_k := \pi_k \bar{\vee} \pi^{(i)}$.

(15) If $k < q$ set $q := k + 1$.

If not all the cells of π_k have been chosen go to step (5).

(16) Set $k := k - 1$.

If $k \geq h - 1$ go to step (13).

Set $h := k + 1$;

$m := m \times |\pi_h(1)|$.

(17) If $k < 0$ *stop*.

Set $\pi_k :=$ partition of $\mathcal{C}(\xi_k, v_k)$ induced from ξ.

If $k < q$ set $q := k + 1$.

Go to step (5).

9.25 Since Algorithm 9.24 produces the same elements of $\Gamma(G)$ as does Algorithm 9.21, Theorem 9.17 will still hold. Given the set of generators Y_0, we can construct the whole group $\Gamma(G)$ if desired, as described in 4.11. However, a certain amount of information about the group can be deduced directly from Y_0. We have seen that the size and the orbits of $\Gamma(G)$ are given by the algorithm. For the next few results we continue the notation of 9.16.

9.26 Theorem: *Suppose* Y_0 *contains an element of the form* $\gamma\delta$ *where* $\gamma, \delta \in \Gamma(G)$ *and any point of* V *not fixed by* γ *is fixed by* δ. *Then either* γ *or* δ *is trivial.*

Proof: Without loss of generality, suppose that for some r where $0 \leq r < \ell$ we have $\nu_r^{\gamma} = \nu_r = \nu_r^{\delta}$ but that $\nu_{r+1}^{\gamma} \neq \nu_{r+1}$. Then $\nu_{r+1}^{\delta} = \nu_{r+1}$ and so $\delta \in \Gamma^{(r+1)}$.

Therefore the terminal node $e_1^{\gamma\delta}$ is in the subtree $T(\nu_{r+1}^{\gamma})$. Let τ be the first terminal node of $T(\nu_{r+1}^{\gamma})$ such that for some $\beta \in \Gamma^{(r+1)}$ which fixes points of V not fixed by γ we have $\tau = e_1^{\gamma\beta}$.

Let $X(\gamma)$ be the set of points fixed by γ. We prove by induction that β is trivial.

Suppose that for some j, where $0 \leq j < \ell$, we have $\nu_j^{\gamma\beta} = \nu_j^{\gamma}$. This is true for example if $j \leq r + 1$.

Then $\mathcal{W}(G, \nu_j^{\gamma\beta}) = \mathcal{W}(G, \nu_j^{\gamma}) = (\mathcal{W}(G, \nu_j))^{\gamma}$ by 8.28. Let $v = v_{j+1}$ for convenience.

(1) If $v \notin X(\gamma)$ then $v^{\beta} = v$ by assumption, and so $v^{\gamma\beta} = v^{\gamma}$, since $\gamma\beta = \beta\gamma$.

(2) If $v \in X(\gamma)$ then

$$v = \min\{\mathcal{W}(G, \nu_j)\} = \min\{\mathcal{W}(G, \nu_j) \cap X(\gamma)\}$$

since $v \in X(\gamma)$.

$$\begin{aligned} \text{Also, } v^{\gamma\beta} &= \min\{(\mathcal{W}(G, \nu_j))^{\gamma} \cap X(\gamma)\} \\ &= \min\{\mathcal{W}(G, \nu_j) \cap X(\gamma)\} \end{aligned}$$

since $w^{\gamma} = w$ if $w \in X(\gamma)$

$$\begin{aligned} &= v \\ &= v^{\gamma}, \text{ since } v \in X(\gamma). \end{aligned}$$

Hence in either case $\nu_{j+1}^{\gamma\beta} = \nu_{j+1}^{\gamma}$, and so $\tau = e_1^{\gamma}$ by induction.

Since the node ν_{r+1}^{γ} will be absorbed onto the node ν_{r+1}, no terminal node of $T(\nu_{r+1}^{\gamma})$ equivalent to e_1 other than e_1^{γ} will be encountered.

Hence $\delta = \beta$ and so δ is trivial. □

9.27 Corollary: *Suppose that for some subset* $Y \subseteq Y_0$ *we have* $\langle Y \rangle = \Psi^{(1)} \oplus \Psi^{(2)}$, *where* $\Psi^{(1)}$ *and* $\Psi^{(2)}$ *are non-trivial subgroups of* $\Gamma(G)$. *Then we can write* $Y = Y^{(1)} \cup Y^{(2)}$ *where* $\langle Y^{(1)} \rangle = \Psi^{(1)}$ *and* $\langle Y^{(2)} \rangle = \Psi^{(2)}$.

Proof: Any element of Y is of the form $\delta\gamma$ where $\gamma \in \Psi^{(1)}$ and $\delta \in \Psi^{(2)}$. By the theorem one of γ and δ is trivial. □

9.28 Theorem: *For some* $\nu \in Q(V)$ *suppose* Γ_ν *has exactly one non-trivial orbit, where* $\Gamma = \Gamma(G)$. *Then there is a subset* $Y^* \subseteq Y_0$ *such that* $\langle Y^* \rangle$ *is conjugate to* Γ_ν *in* Γ.

Proof: For any subgroup $\Lambda \leq \Gamma$ let $\chi(\Lambda)$ denote the set of points fixed by Λ and let $\ell(\Lambda)$ denote the maximum value of j for which $\{v_1, \cdots, v_j\} \subseteq \chi(\Lambda)$.

Let Ψ be a subgroup of $\Gamma(G)$ conjugate to Γ_ν for which $r = \ell(\Psi)$ is the greatest. Let C be the non-trivial orbit of Ψ.

By assumption, $v_{r+1} \in C$. Also, since $\Psi \leq \Gamma^{(r)}$, C is contained in some orbit C_1 of $\Gamma^{(r)}$.

Suppose there exists a point v in $C_1 \backslash C$. Then if $\gamma \in \Gamma^{(r)}$ and $v_{r+1} = v^{\gamma}$, $\gamma^{-1}\Psi\gamma$ is in $\Gamma^{(r)}$ and fixes v_{r+1}. This contradicts the maximality of $\ell(\Psi)$, and so $C = C_1$.

Now suppose C_2 is another orbit of $\Gamma^{(r)}$ and $w \in C_2$. Since the partition θ_{Ψ} is equitable (5.9) and fixes w, the cell {w} is trivially joined to the cell C.

However, the equitable partition $\theta_{\Gamma^{(r)}}$ also has C as a cell, and so C_2 is trivially joined to C.

Hence by 5.16, $\Gamma^{(r)} = \Gamma^{(r)}|_C \oplus \Gamma^{(r)}|_{V\backslash C}$ and so by 9.27, Y_0 contains a subset generating $\Gamma^{(r)}|_C = \Psi$. □

9.29 Corollary: *If* $\Gamma(G)$ *contains transpositions, then* Y_0 *contains at least one member from each conjugacy class of transpositions.*

Proof: The subgroup of $\Gamma(G)$ generated by a single transposition satisfies the conditions of the theorem. □

It is not clear when Theorem 9.28 will hold without the restriction that Γ_ν have just one non-trivial orbit. We conjecture that a sufficient condition is that for any $v \in V$ not fixed by Γ_ν the stabliser $(\Gamma_\nu)_v$ is trivial.

9.30 Theorem: *If* $\Gamma(G) = \Psi[\Sigma]$, *where* Ψ *and* Σ *are non-trivial, then* Y_0 *contains a subset generating one of the copies of* Σ *in* $\Gamma(G)$.

Proof: For subgroups $\Lambda \in \Gamma(G)$ define $\ell(\Lambda)$ as before and let Σ^* be the copy of Σ for which $r = \ell(\Sigma^*)$ is greatest. Then $\Gamma^{(r)}$ is a direct

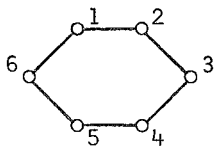
sum with one factor Σ^*. The result follows from 9.27. □

9.31 We now give a sequence of examples of the performance of Algorithms 9.21 and 9.24. The following abbreviations are used:

n	:	number of points of G.
ν_ℓ	:	first ancestor of e_1 for which $\mathcal{B}(G, \nu_\ell)$ is discrete.
M_1	:	number of terminal nodes equivalent to e_1.
M_2	:	number of terminal nodes not equivalent to e_1.
ξ	:	orbits partition of $\Gamma = \Gamma(G)$.
π_0	:	unit partition of V.
t	:	execution time in milliseconds, excluding time for output.

If both 9.21 and 9.24 behave the same way and take about the same time, only the figures for 9.21 are given.

9.32



n = 6.

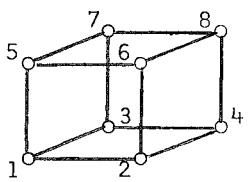
ν_ℓ = [1, 2]; M_1 = 3, M_2 = 0.

$|\Gamma|$ = 12; $\xi = \pi_0$.

Y_0 = {(2 6)(3 5), (1 2)(3 6)(4 5)}.

t = 5·8.

9.33



n = 8.

ν_ℓ = [1, 2, 3]; $M_1 = 4$, $M_2 = 0$.

$|\Gamma| = 48$; $\xi = \pi_0$.

Y_0 = {(3 5)(4 6), (2 3)(6 7), (1 2)(3 4)(5 6)(7 8)}.

t = 10•3.

9.34 $G = K_{10}$.

n = 10.

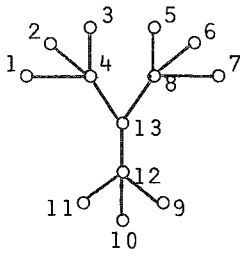
ν_ℓ = [1, 2, 3, 4, 5, 6, 7, 8, 9]; $M_1 = 10$, $M_2 = 0$.

$|\Gamma| = 3{,}628{,}800$; $\xi = \pi_0$.

Y_0 = {(9 10), (8 9), (7 8), (6 7), (5 6), (4 5), (3 4), (2 3), (1 2)}.

t = 36•4.

9.35



n = 13.

ν_ℓ = [4, 8, 1, 2, 9, 10, 5, 6]; $M_1 = 9$, $M_2 = 0$.

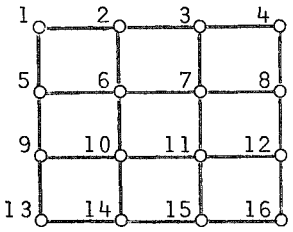
$|\Gamma| = 1296$; ξ = {1, 2, 3, 5, 6, 7, 9, 10, 11|4, 8, 12|13}.

Y_0 = {(6 7), (5 6), (2 3), (1 2), (10 11), (9 10),

(5 9)(6 10)(7 11)(8 12), (1 5)(2 6)(3 7)(4 8)}.

t = 38•6.

9.36



n = 16.

ν_ℓ = [1, 12]; M_1 = 3, M_2 = 0.

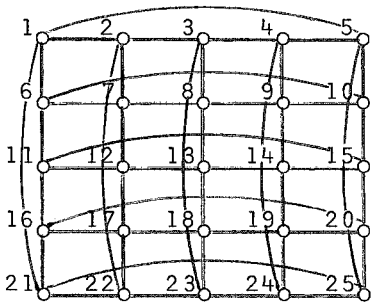
$|\Gamma|$ = 8; ξ = {1, 4, 13, 16|2, 3, 5, 8, 9, 12, 14, 15|6, 7, 10, 11}.

Y_0 = {(2 5)(3 9)(4 13)(7 10)(8 14)(12 15),

(1 4)(2 3)(5 8)(6 7)(9 12)(10 11)(13 16)(14 15)}.

t = 21·5.

9.37



n = 25.

ν_ℓ = [1, 13, 5]; M_1 = 4, M_2 = 0.

$|\Gamma|$ = 200; $\xi = \pi_0$.

Y_0 = {(2 6)(3 11)(4 16)(5 21)(8 12)(9 17)(10 22)(14 18)(15 23)(20 24),

(2 5)(3 4)(7 10)(8 9)(12 15)(13 14)(17 20)(18 19)(22 25)(23 24),

(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)}.

t = 60·9.

9.38 G = $C_5[C_5]$, where C_5 is labelled in a circular fashion. The group $\Gamma(G)$ is $\Psi[\Psi]$, where $\Psi = \Gamma(C_5)$, [58].

$n = 25.$

$\nu_\ell = [1, 3, 11, 13, 21, 23, 16, 18, 6, 8]$; $M_1 = 13$, $M_2 = 0$.

$|\Gamma| = 1{,}000{,}000$; $\xi = \pi_0$.

$Y_0 = \{(7\ 10)(8\ 9),\ (6\ 7\ 8\ 9\ 10),\ (17\ 20)(18\ 19),$

$(16\ 17\ 18\ 19\ 20),\ (22\ 25)(23\ 24),\ (21\ 22\ 23\ 24\ 25),$

$(12\ 15)(13\ 14),\ (11\ 12\ 13\ 14\ 15),$

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

$(2\ 5)(3\ 4),\ (1\ 2\ 3\ 4\ 5),$

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

$t = 160.$

9.39 G is the graph with points $\{1, 2, \cdots, 13, 1', 2', \cdots, 13'\}$ where for $1 \le i \le 13$, $1 \le j \le 13$,

i and j are adjacent iff $i - j = 2, 5, 6, 7, 8$ or 11,

i' and j' " " " " $= 1, 3, 4, 9, 10$ or 12,

i and j' " " " " $= 0, 1, 3$ or 9,

all differences being taken modulo 13, [1].

$n = 26.$

$\nu_\ell = [1, 3]$; $M_1 = 3$; $M_2 = 5$ (for 9.24), 7 (for 9.21).

$|\Gamma| = 39$; $\xi = \{1, 2, \cdots, 13|1', 2', \cdots, 13'\}$.

$Y_0 = \{\alpha\alpha', \beta\beta'\}$

where $\alpha = (2\ 10\ 4)(3\ 6\ 7)(5\ 11\ 13)(8\ 12\ 9)$,

$\beta = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13)$,

and α', β' denote the corresponding permutations acting on the second half of G.

$t = 99$ (for 9.24), 116 (for 9.21).

9.40 G is the strongly regular graph with degree 10, 26 points and trivial automorphism group, as given in [65].

$n = 26.$

$\nu_\ell = [1, 17, 7]$; $M_1 = 1$, $M_2 = 7$ (for 9.24), 267 (for 9.21).

$|\Gamma| = 1$; $Y_0 = \phi$.

$t = 1{\cdot}15$ seconds (for 9.24), $2{\cdot}60$ seconds (for 9.21).

The algorithm of Arlazarov et al. [2] produced 756 terminal nodes for this graph, and 40 for the graph of 9.39. They do not state their execution times.

9.41 We now consider the efficiency of Algorithm 9.21. Although Algorithm 9.24 is always at least as fast as 9.21, we have not been able to find any simple estimates for its execution time which are better than those for 9.21. Furthermore, we have not been able to estimate the effect of the improvement described in 9.20 although, as we have said, it is often considerable. Consequently, we will assume that at step (2) we always have $n - |\xi_k| > 5$.

Define $M = M_1 + M_2$, where M_1 and M_2 are as defined in 9.31. Let t be the total time taken by Algorithm 9.21 when applied to $G \in \underset{\sim}{G}(V)$.

9.42 We first consider the time t_1 taken for the computation of $\mathcal{B}(G, \nu)$ for each node ν. Let $\tau = [v_1, \cdots, v_n]$ be a terminal node. For $0 \leq j \leq n$ define $\nu_j = [v_1, \cdots, v_j]$,

$$\pi_j = \mathcal{B}(G, \nu_j),$$

$$\ell_j = |\pi_j|.$$

Let k be the smallest value of j for which $\ell_j = n$.

By definition (9.7),

$$\pi_{j+1} = \mathcal{R}_{\ell_j+1}(G, \mathcal{D}(\pi_j, v_{j+1})), \quad \text{for} \quad 0 \leq j < k.$$

The computation of $\mathcal{D}(\pi_j, v_{j+1})$ requires time of order 1, and so the computation of π_{j+1} takes time of order $n(\ell_{j+1} - \ell_j)$, by 5.40. Similarly the computation of π_0 requires time of order $n\ell_0$.

Hence the time taken to compute π_j for ν_k and its ancestors is of order

$$n\ell_0 + \sum_{j=0}^{k-1} n(\ell_{j+1} - \ell_j) = n\ell_0 + n(\ell_k - \ell_0)$$
$$= n\ell_k$$
$$= n^2.$$

Summing over all terminal nodes we have

$$t_1 = O(n^2M).$$

9.43 Next we consider the time t_2 required for calculations of the form $\pi_k := \pi_k \bar{\vee} \pi^{(i)}$ at step (12). One such computation takes time of order $|\pi_k|\omega(\pi^{(i)})$ where $\omega(\pi^{(i)})$ is the number of non-trivial cells of $\pi^{(i)}$. Define

$$\Omega = \sum_{i=1}^{m} \omega(\pi^{(i)}), \text{ where } m = M_1 - 1.$$

Then t_2 is of order $\Omega\Sigma|\pi_k|$, where the sum is taken over all ancestors of terminal nodes not equivalent to e_1. We conjecture that for any n, $\Omega \leq \frac{1}{2}n\{\log_2 n\}$, where $\{\log_2 n\}$ is the smallest integer not smaller than $\log_2 n$. This bound has been proven for graphs whose automorphism groups Γ have the property that, for any $v \in V$ not fixed by Γ, the stabiliser Γ_v is trivial, and for a few other similar cases. However, the best bound we have been able to prove for an arbitrary graph is $\Omega \leq \frac{n}{8}(3n - 2)$. Hence the best we can say for certain is that t_2 is of order n^4M_2, although no class of graphs has been found for which an order worse than n^2M_2 holds.

9.44 Other contributions to the execution time of 9.21 are easily bounded. We list them below.

(i) For the computation of $\mathcal{C}(\xi_k, \nu_k)$ at step (3): $O(n^2M)$.

(ii) For computing the adjacency matrix of $G(\xi_k)$ at step (5): $O(n^2M)$.

(iii) For comparison of $G(\xi_k)$ with $G(\varepsilon)$ and $G(\rho)$: $O(nM)$.

(iv) For computing γ, θ_γ, $\xi \bar{\vee} \theta_\gamma$ and $\pi_k \bar{\vee} \theta_\gamma$ at step (8): $O(n^2M_1)$.

(v) For setting π_k at step (15): $O(n^2)$.

(vi) For indexing and other minor computations: $O(n^2M)$.

Most of these bounds follow from the fact that T_G has M terminal nodes and not more than nM + 1 nodes. Bound (iv) follows from the observation that γ is only computed for terminal nodes equivalent to e_1.

9.45 Putting these estimates together, we find that the total time t is at worst of order $n^2M_1 + n^4M_2$, although, as stated in 9.43, we know of no class of graphs for which $t > O(n^2M_1 + n^2M_2) = O(n^2M)$.

By Theorem 9.17, $M_1 \leq n$ and so $t = O(n^3 + n^4M_2)$. No realistic estimate for M_2 has been found, since it depends on two factors, both of them difficult to determine.

(i) The number of identity nodes depends on the relationship between $\Theta(G)$ and $\Xi(G)$.

(ii) The efficiency of the technique of 7.23 in reducing the number of terminal nodes equivalent to identity nodes other than e_1

is difficult to estimate. In fact, it depends on the labelling of G.

9.46 Fortunately, the proportion of graphs for which $M_2 > 0$ is quite small. For graphs with 7, 8 or 9 points the proportions are respectively 2/1044, 15/12346 and 70/274668 and in no graph with ≤ 9 points have we observed $M_2 > 5$.

9.47 Theorem: *The following condition is sufficient to ensure that* $M_2 = 0$ *for* G.

For any $\nu \in Q(V)$, *let* π *be the coarsest element of* $\Xi(G)$ *which fixes* ν. *Then the non-trivial cells of* π *of smallest size are orbits of* Γ_ν, *where* $\Gamma = \Gamma(G)$.

Proof: From 9.9 we have $\mathcal{B}(G, \nu) \simeq \pi$. The result follows from 9.3 (2) and the definition of $\mathcal{W}$. □

9.48 Corollary: *If* G *is s-e,* $M_2 = 0$.

Proof: If G is s-e, then all equitable partitions are orbital. □

Unfortunately, the conditions of 9.47 and 9.48 are both very difficult to verify, both theoretically and experimentally. Incidentally we do not know of any transitive graph for which $M_2 \neq 0$.

Let $M_2(n)$ be the maximum value of M_2 for any graph with n points. The following result shows that $M_2(n)$ is not bounded above by any polynomial in n.

9.49 Theorem: *Let* G *be a connected regular graph with* m *points, whose*

automorphism group $\Gamma(G)$ *has* p *orbits. Then for any* $k > 0$, $M_2(km) \geq p^k - 1$.

Proof: Let kG denote the graph consisting of k disjoint copies of G, $\{G_1, \cdots, G_k\}$ with point sets $\{V_1, \cdots, V_k\}$ respectively, and define $V = V_1 \cup \cdots \cup V_k$.

(a) Suppose $k = 1$.

Since G_1 is regular, $\mathcal{W}(G_1, [\,]) = V_1$, and so contains p orbits of $\Gamma(G_1)$. Hence T_{G_1} has at least p identity nodes.

(b) Suppose T_{rG} has at least p^r identity nodes, for some $r > 0$. Let $H = (r + 1)G$. Since H is regular, $\mathcal{W}(H, [\,]) = V$ and so contains p orbits of $\Gamma(H)$. Hence T_H has p equivalence classes (under $\Gamma(H)$) of nodes of length one.

Suppose $\nu_1 = [v] \in T_H$ is the first node in one of these classes. Without loss of generality we can assume that $v \in V_1$. By 9.9, $\mathcal{B}(H, \nu_1)$ contains one cell $C = V_2 \cup \cdots \cup V_{r+1}$. All its other cells are proper subsets of V_1, and hence smaller than C. If any of these cells is non-trivial, we have $\mathcal{W}(H, \nu_1) \subseteq V_1$, by the definition of $\mathcal{C}$.

Continuing in this manner down the subtree $T_H(\nu_1)$, we eventually find a node ν_j for which $\mathcal{B}(H, \nu_j)$ contains exactly one non-trivial cell, namely $V_2 \cup \cdots \cup V_{r+1}$. Since the trivial cells of $\mathcal{B}(H, \nu_j)$ will have no further effect on the computation of $\mathcal{W}(H, \cdot)$ for nodes of $T_H(\nu_j)$, we can apply the induction hypothesis and say that $T_H(\nu_j)$, and hence $T_H(\nu_1)$, has at least p^r identity nodes. Considering the other equivalence classes of unit-length nodes, we see that T_H has at least p^{r+1} identity nodes. □

9.50 We now give some experimental data on the performance of Algorithms 9.21 and 9.24 for a few common families of graphs. The execution times are shown in Figure 9.4, where both scales are logarithmic. The approximate gradient of the curve for large n is given below as the constant κ. As before, only data for 9.21 is given, unless 9.24 behaves appreciably different.

(a) The path on n points, P_n, labelled from one end to the other.

For all n, $M_1 = 2$ and $M_2 = 0$. $\kappa \simeq 1{\cdot}8$.

(b) The cycle on n points, Z_n, labelled in a circular fashion.

For all n, $M_1 = 3$ and $M_2 = 0$. $\kappa \simeq 1{\cdot}9$.

(c) The complete graph on n points, K_n.

For each n, $M_1 = n$ and $M_2 = 0$. $\kappa \simeq 2{\cdot}7$.

(d) The generalised cube on 2^m points, Q_m, defined by $Q_1 = P_2$, $Q_m = Q_{m-1} \times P_2 \, (m > 1)$.

For each m, $M_1 = m + 1$ and $M_2 = 0$. $\kappa \simeq 2{\cdot}0$.

(e) Random graphs, as defined in 3.11.

The two curves marked RG in Figure 9.4 show average execution times for $\sigma = 0{\cdot}50$ and $\sigma = 0{\cdot}75$. In both cases, no graphs with non-trivial automorphism groups were encountered for $n > 25$, and no graphs for which $M_2 \neq 0$ were encountered for $n \geq 10$. Hence the measured times for 9.21 and 9.24 were almost identical.

To illustrate how fast these algorithms are on random graphs we have plotted (as a dashed line in Figure 9.4) the time required for a single permutation of an $n \times n$ adjacency matrix. For $n = 30$ and $n = 60$ this time represents about 58% and 73%, respectively, of the time taken by 9.21 on random graphs. Since at least one such matrix

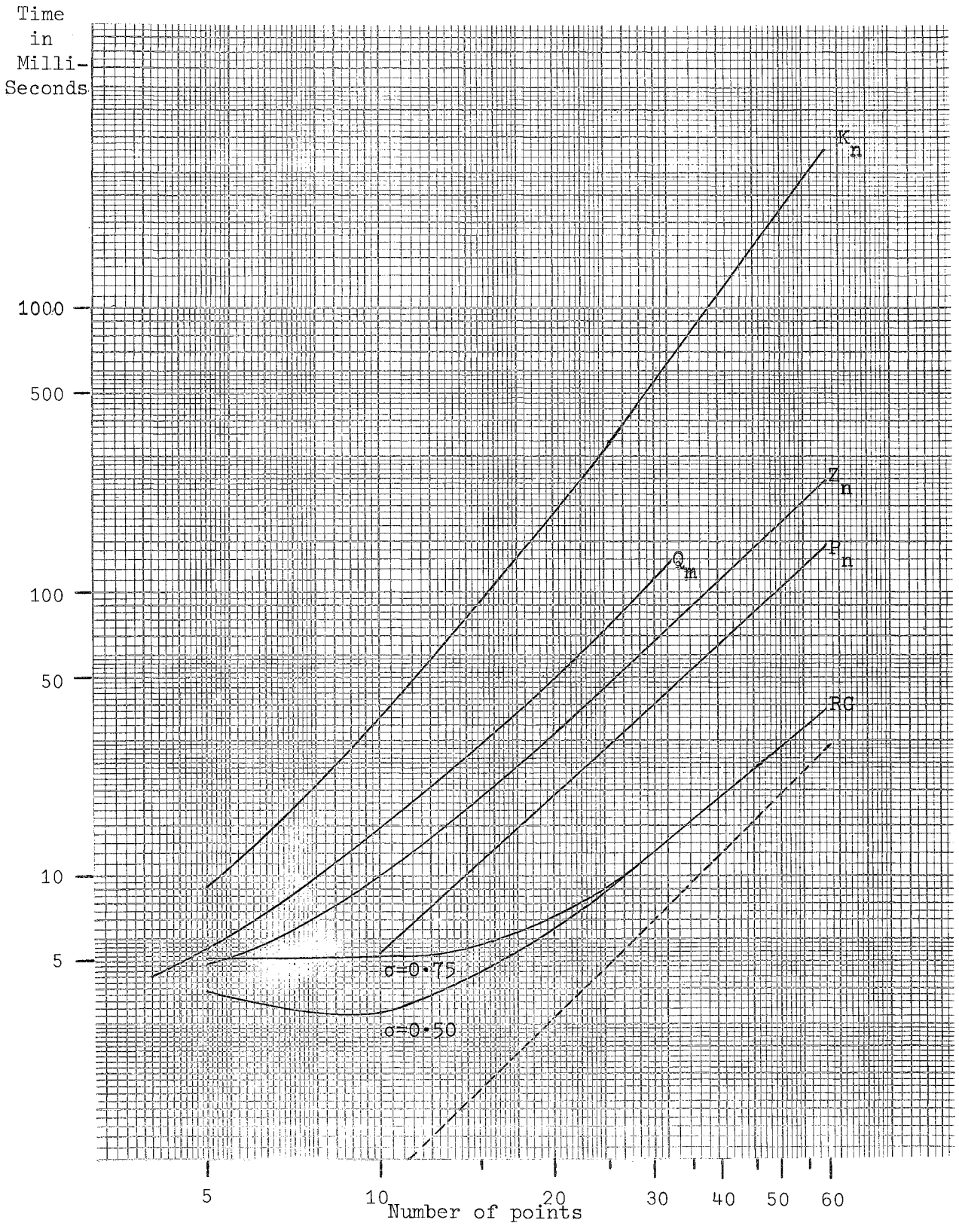

Time in Milli-Seconds
1000
500
100
50
10
5
K_n
Z_n
P_n
Q_m
RG
σ=0·75
σ=0·50
5
10
20
30
40
50
60
Number of points

Figure 9.4

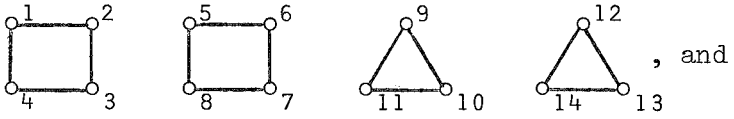
permutation is an essential step for any canonical labelling algorithm which employs an adjacency matrix representation of a graph, we believe that it is not possible to devise such an algorithm which is very much faster than our own on large random graphs.

9.51 Only a few other authors have given execution times for their algorithm's performance on random graphs. In Table 9.1 we list the execution times (in seconds) of Algorithm 9.21, Corneil and Gotlieb's algorithm [11, 14] and Ullmann's algorithm [75], for random graphs with $\sigma = 0{\cdot}5$. Both these other algorithms test for isomorphism between two graphs. It is clear that Algorithm 9.21 is by far the fastest, even after allowing for machine-speed differences (perhaps a factor of 4 in both cases). Times marked with a dagger (†) in Table 9.1 were estimated from related figures given by the relevant authors.

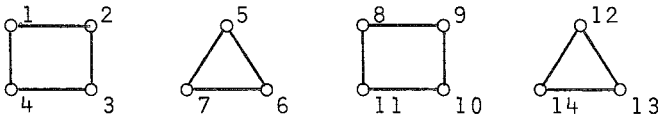
n	9·21	Corneil and Gotlieb	Ullmann
20	0·0065	0·27	0·90
40	0·020	$0{\cdot}95^{(\dagger)}$	$6{\cdot}1^{(\dagger)}$
60	0·039	1·98	$19^{(\dagger)}$

Table 9.1

9.52 Let H_k denote the graph with 2k components, k isomorphic to Z_3 and k isomorphic to Z_4. We define $H_k^{(1)}$ and $H_k^{(2)}$ to be particular labelled graphs isomorphic to H_k. In $H_k^{(1)}$ all the copies of Z_4 are labelled before the copies of Z_3, and in $H_k^{(2)}$ copies of Z_4 and Z_3 are labelled alternately. For example,

$H_2^{(1)}$ = 

, and

$H_2^{(2)}$ = 

.

For any k, the program tree T_{H_k} has $\binom{2k}{k}$ identity nodes, and so we can expect Algorithms 9.21 and 9.24 to be comparatively inefficient on these graphs. However, for both labellings, Algorithm 9.24 finds no identity nodes not equivalent to e_1, showing that the technique described in 9.23 has been very successful. The behaviour of Algorithm 9.21 can be seen from Table 9.2. M_1 was the same for both labellings.

k	n	$\lvert\Gamma(H_k)\rvert$	$\binom{2k}{k}$	M_1	M_2 for $H_k^{(1)}$	M_2 for $H_k^{(2)}$
1	7	48	2	5	1	1
2	14	9216	6	11	5	10
3	21	3981312	20	17	19	57
4	28	$3{\cdot}06 \times 10^{9}$	70	23	69	276
5	35	$3{\cdot}67 \times 10^{12}$	252	29	251	1257
6	42	$6{\cdot}34 \times 10^{15}$	924	35	923	5555
7	49	$1{\cdot}49 \times 10^{19}$	3432	41	3431	24000(†)
8	56	$4{\cdot}58 \times 10^{22}$	12870	47	12869	104000(†)

[(†)estimated]

Table 9.2

The reason why M_2 is larger for $H_k^{(2)}$ than for $H_k^{(1)}$ seems to be that terminal nodes not equivalent to e_1 are encountered before very many elements of $\Gamma(H_k)$ have been found, so that the process in step (12) of 9.21 is not so effective. However, for k = 8 we still have only about 8 terminal nodes per identity node, which is *very* small compared with $|\Gamma(H_k)|$. Execution times for both algorithms, and both labellings, are shown in Figure 9.5.

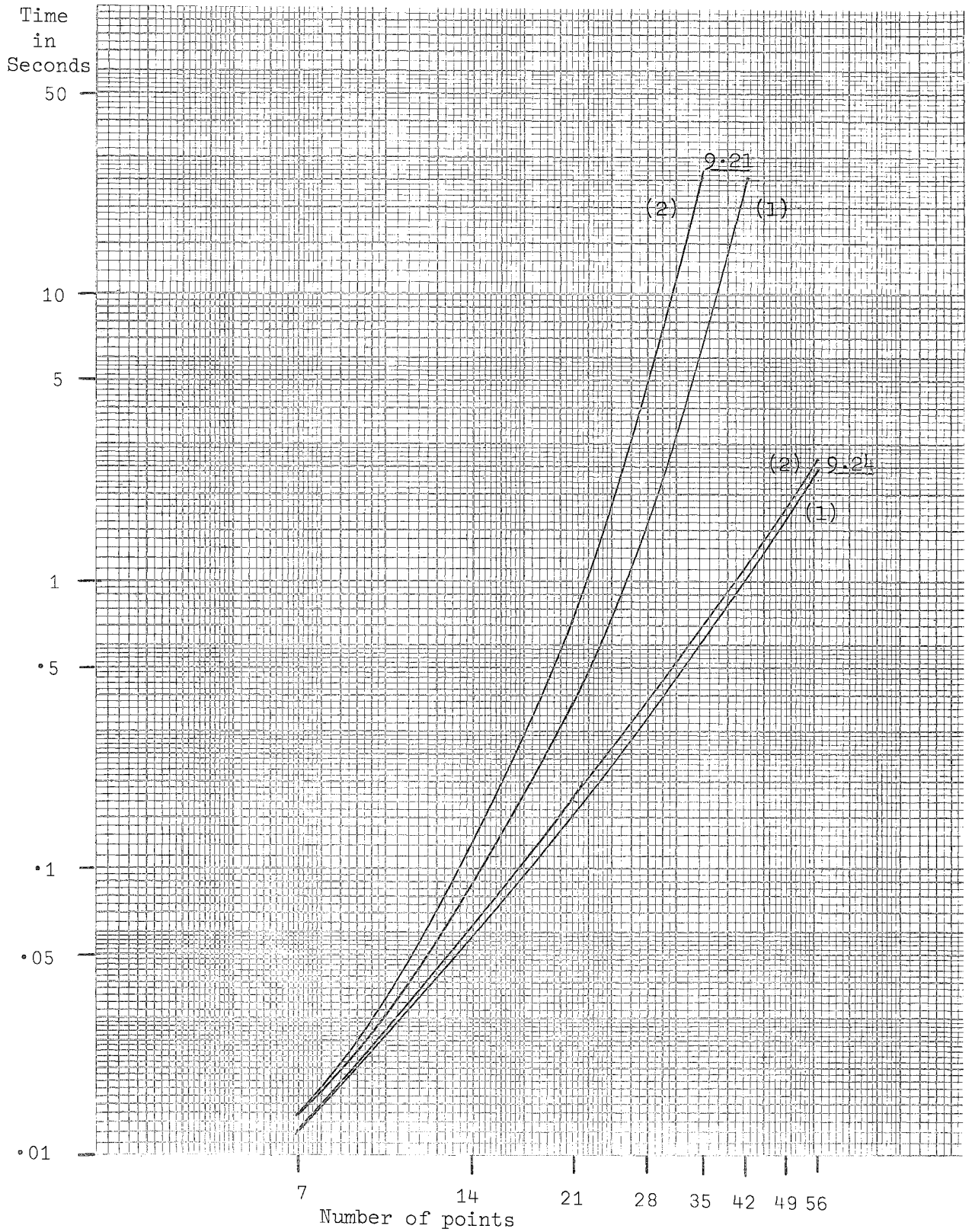

Time in Seconds
50
10
5
1
·5
·1
·05
·01
9·21
(2)
(1)
(2) 9·24
(1)
7
14
21
28
35
42
49
56
Number of points

Figure 9.5

9.53 A minor extension of Algorithms 9.21 and 9.24 enables them to solve two somewhat more general problems. Suppose $G \in \underset{\sim}{G}(V)$ and $\zeta \in \widetilde{\Pi}(V)$. One problem is the determination of Γ_ζ, where $\Gamma = \Gamma(G)$. The other is to find a map

$$f : \underset{\sim}{G}(V) \times \widetilde{\Pi}(V) \to \underset{\sim}{G}(V)$$

so that the following hold for each $G \in \underset{\sim}{G}(V)$, $\zeta \in \widetilde{\Pi}(V)$ and $\gamma \in S_n$.

(1) $f(G, \zeta)$ is isomorphic to G.

(2) $f(G^\gamma, \zeta^\gamma) = f(G, \zeta)$.

(3) $f(G, \zeta^\gamma) = f(G, \zeta)$ iff $\zeta^\gamma = \zeta^\delta$ for some $\delta \in \Gamma(G)$.

Clearly,this definition generalises that of a canonical labelling as given in 8.5. We can think of it as the problem of canonically labelling a graph with coloured points, each colour corresponding to a cell of ζ.

Although we shall not prove it here, the only change required to 9.21 and 9.24 is to alter 9.7(1) to read

"If $|\nu| = 0$, define $\mathcal{B}(G, \nu) = \mathcal{R}_1(G, \zeta)$."

9.54 A particular application of this technique can be described as follows. Suppose $G, H \in \underset{\sim}{G}(V)$ and that G and H are known to be transitive. If G and H are isomorphic and $v \in V$, there is an isomorphism from G to H which takes $v \in V(G)$ onto $v \in V(H)$. Therefore we can compare G and H by using Algorithm 9.21 with $\zeta = [v|V\backslash\{v\}]$. This will generally save a considerable amount of time. The elements of $\Gamma(G)$ found by 9.21 (or 9.24) will generate the stabiliser $\Gamma(G)_v$.

In one of the first practical applications of Algorithm 9.21, this method was used to generate all the circulant graphs with fewer than 38 points. A graph G with n points is *circulant* if $\Gamma(G)$ contains a cycle of length n; hence G is transitive. In one run, for example, the isomorphic copies amongst 23423 circulant graphs with 36 points were found in less than 30 minutes. For these graphs, M_2 was always zero, and M_1 averaged about 2·4.

9.55 Algorithms 9.21 and 9.24 can also be easily extended so that they apply to more general graph-like objects, for example digraphs, loop-graphs or multigraphs. We have used 9.21 and 9.24 with considerable success on both digraphs and loop-graphs. The only necessary change was to suspend the technique described in 9.20, since Theorem 5.19 no longer holds.

9.56 Finally, we mention a few simple methods by which our algorithms might be improved. Basic directions we might try to take are towards reducing the number of identity nodes, and towards reducing the number of non-identity nodes.

Considering the first possibility, suppose $G \in \underset{\sim}{G}(V)$ and that ω_1 and ω_2 are maps satisfying 8.11. If r_1 and r_2 are the number of identity nodes of the program trees defined by $\omega_1(G, \cdot)$ and $\omega_2(G, \cdot)$ respectively, we say that ω_1 is *stronger* than ω_2 if $r_1 \leq r_2$. If $r_1 = 1$, then ω_1 is *optimal* (for G). The well-known Corneil-Gotlieb algorithm [14] uses a defining function stronger than ours, but requiring much more time for its evaluation. In fact, these authors conjecture their choice of ω to be always optimal, but unfortunately counter-examples have since been found [13]. On the other hand, Arlazarov et al. [2] use a defining function which can be very rapidly

evaluated, but which is weaker than ours. The maps used by Overton and Proskurowski [49] probably also fall into this category. We believe that for most graphs our own choice of $\mathcal{W}$ is a reasonable compromise, since it is fast to compute (5.41) and usually optimal (9.46). However, to help those cases (like the graphs in 9.52) where $\mathcal{W}$ is far from optimal, it should be possible to devise a system whereby a stronger version of $\mathcal{W}$ is automatically "turned on" by the appearance of too many identity nodes. Even if the algorithm must be restarted in these cases (which is not certain), this system should substantially improve the "worst-case" behaviour without damaging the average efficiency. For this purpose, we are currently examining several possible choices for a map c (or a sequence of maps) satisfying 8.23. The idea is to apply Algorithm 5.32 first on c(G) and then on G during the computation of $\mathcal{B}$.

9.57 The other possibility for improvement could be to reduce the number of non-identity terminal nodes. Since the number equivalent to e_1 is already very small (9.17), these nodes would no longer be a problem if the number of identity nodes was sufficiently reduced. Nevertheless there may be some merit in having $L > 0$, as described in 7.27.

BIBLIOGRAPHY

Papers not referred to in the text are indicated by an asterisk.

[1] G. ADEL'SON-VEL'SKII, B. VEISFEILER, A. LEMAN and I. FARADZEV: Example of a graph without a transitive automorphism group. *Dokl. Akad. Nauk SSSR* 185, 5 (1969). Translation: *Soviet Math. Dokl.* 10, 2 (1969) 440-441.

[2] V. ARLAZAROV, I. ZUEV, A. USKOV and I. FARADZEV: An algorithm for the reduction of finite non-oriented graphs to canonical form. *Zh. vȳchisl. Mat. mat. Fiz.* 14, 3 (1974) 737-743.

[3] H. BAKER, A. DEWDNEY and A. SZILARD: Generating the nine-point graphs. *Math. Comp.* 28, 127 (1974) 833-838.

[4] M. BEHZAD and G. CHARTRAND: "Introduction to the Theory of Graphs". Allyn and Bacon, Boston (1971).

[5] A. BERZTISS: A backtrack procedure for isomorphism of directed graphs. *JACM* 20, 3 (1973) 365-377.

[6] N. BIGGS: "Algebraic Graph Theory". Cambridge Tracts in Mathematics No. 67 (1974).

[7] G. BIRKHOFF: "Lattice Theory". 3rd edition. Providence, Rhode Island, AMS (1973).

*[8] C. BOHM and A. SANTOLINI: A quasi-decision algorithm for the P-equivalence of two matrices. *ICC Bull.* 3 (1964) 57-69.

*[9] H. BROWN and L. MASINTER: An algorithm for the construction of the graphs of organic molecules. *Discrete Maths.* 8 (1974) 227-244.

[10] L. COLLATZ and U. SINOGOWITZ: Specktren endlicher Grafen. *Abh. Math. Sem. Univ. Hamburg* 21 (1957) 63-77.

[11] D.G. CORNEIL: "Graph Isomorphism". Ph.D. Thesis, Univ. of Toronto (1968).

*[12] D.G. CORNEIL: An algorithm for determining the automorphism partitioning of an undirected graph. *BIT* 12 (1972) 161-171.

[13] D.G. CORNEIL: The analysis of graph theoretic algorithms. *Univ. of Toronto, Dept. of Computer Sci., Technical Report* No. 65 (1974).

[14] D.G. CORNEIL and C.C. GOTLIEB: An efficient algorithm for graph isomorphism. *JACM* 17, 1 (1970) 51-64.

[15] D.Z. DJOKOVIC: Automorphisms of graphs and coverings. *J. Comb. Th. (B)* 16 (1974) 234-247.

*[16] J.P. DOLCH: Names and aliases of graphs. *Proc. 3rd S-E Conf. on Combinatorics, Graph Theory and Computing* (1972) 175-194.

[17] A. DUIJVESTIJN: Electronic computation of squared rectangles. *Philips Res. Rep.* 17 (1962) 537.

[18] J. FILLMORE and S. WILLIAMSON: On backtracking: a combinatorial description of the algorithm. *SIAM J. Comput.* 3, 1 (1974) 41-55.

[19] H. FINCK and H. SACHS: Uber Beziehungen zwischen Struktur und Spektrum regulärer Graphen. *Wis. Z. Th. Ilmenau* 19 (1973) 83-99.

[20] I. FLORES: "Computer Sorting". Prentice-Hall, N.J. (1969).

[21] C. GODSIL and B. McKAY: Some computational results on the spectra of graphs. *Proc. 4th Australian Conf. on Combinatorial Mathematics, Adelaide* (1975). To appear.

[22] S.W. GOLOMB and L.D. BAUMERT: Backtrack programming. *JACM* 12, 4 (1965) 516-524.

[23] E.V. HAYNSWORTH: Applications of a theorem on partitioned matrices. *J. Res. Nat. Bur. Stand. (B)* 63 (1959) 73-78.

[24] R.B. HEAP: The production of graphs by computer, in "Graph Theory and Computing", R.C. Read (ed.). Academic Press, N.Y. & Lond. (1972) 47-62.

*[25] J. HOPCROFT and R. KARP: An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput.* 2, 4 (1973) 225-231.

*[26] J. HOPCROFT and R. TARJAN: A V^2 algorithm for determining isomorphism of planar graphs. *Info. Process. Lttrs.* 1, 1 (1971) 32-34.

*[27] J. HOPCROFT and R. TARJAN: Isomorphism of planar graphs, in "Complexity of Computer Computations", R.E Miller and J.W. Thatcher (ed.). Plenum (1972) 131-152.

*[28] J. HOPCROFT and J.K. WONG: Linear time algorithm for isomorphism of planar graphs. (Extended abstract). *Sixth annual ACM symposium on Theory of Computing.* Seattle (1974).

[29] D.B. JOHNSON: Finding all the elementary circuits of a directed graph. *SIAM J. Comput.* 4, 1 (1975) 77-84.

*[30] V. KARELIN and B. MIRONOV: An algorithm for determining an isomorphism of a homogeneous non-oriented graph. *Engng. Cybernetics* 13, 2 (1975) 117-121.

[31] C.A. KING: "A Graph-Theoretic Programming Language". Ph.D. Thesis, Univ. of West Indies (1970).

[32] D.G. KIRKPATRICK: Topics in the complexity of combinatorial algorithms. *Univ. of Toronto, Dept. of Computer Sci., Technical Report* No. 74 (1974).

*[33] W. KNÖDEL: Ein Verfahren zur Festellung der Isomorphie von endlichen, zusammenhängenden Graphen. *Computing* 8 (1974) 329-334.

[34] D.E. KNUTH: Estimating the efficiency of backtrack programs. *Stanford Univ., Dept. of Computer Sci.* STAN-CS-442 (1974).

[35] L. KRIDER: A flow analysis algorithm. *JACM* 11, 4 (1964) 429-436.

[36] W.W. KUHN: "An Algorithm for Graph Isomorphism Using Adjacency Matrices". Ph.D. Thesis, Univ. of Pennsylvania (1971).

[37] W.W. KUHN: A random graph generator. *Proc. 3rd S-E Conf. on Combinatorics, Graph Theory and Computing* (1972) 311-313.

[38] P. LANCASTER: "Theory of Matrices". Academic Press, N.Y. & Lond. (1969).

[39] G. LEVI: A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *CALCOLO* 9 (1972) 341-352.

[40] G. LEVI: Graph isomorphism: a heuristic edge-partitioning-oriented algorithm. *Computing* 12 (1974) 291-313.

[41] G. LEVI and F. LUCCIO: A technique for graph embedding with contraints on node and arc correspondences. *Information Sciences* 5 (1973) 1-23.

[42] M.F. LYNCH: Storage and retrieval of information on chemical structures by computer. *Endeavour* 27, 101 (1968) 68-73.

[43] M.F. LYNCH, J.M. HARRISON, W.G. TOWN and J.E. ASH: "Computer Handling of Chemical Structure Information". Macdonald/ American Elsevier Computer Monographs No. 13 (1971).

[44] M. MASUYAMA: On a test for isomorphism of linear graphs associated with experimental designs. *Sankhyā A* 36, 1 (1974) 53-62.

[45] D.W. MATULA: An algorithm for subtree identification. *SIAM 1967 National Meeting*. (Abstract in *SIAM Review* 10, 2 (1968) 273-274).

[46] H.L. MORGAN: The generation of a unique machine description for chemical structures. *J. Chem. Docum.* 5 (1965) 107-113.

*[47] R.MORPURGO: Un metodo euristico per la verifica dell'isomorfismo di due grafi semplici non orientati. *CALCOLO* 8 (1971) 1-31.

[48] J. NAGLE: On ordering and identifying undirected linear graphs. *J. Math. Phys.* 7 (1966) 1588-1592.

[49] M. OVERTON and A. PROSKUROWSKI: Finding the maximal incidence matrix of a large graph. *Stanford Univ., Dept. of Computer Sci.* STAN-CS-509 (1975).

[50] R. PARRIS: "The Coding Problem for Graphs". M.Sc. Thesis, Univ. of West Indies (1968).

[51] R. PARRIS and R.C. READ: A coding procedure for graphs. *Scientific Report*. UWI/CC 10. *Univ. of West Indies Computer Centre* (1969).

[52] R.H. PENNY: A connectivity code for use in describing chemical structures. *J. Chem. Docum.* 5 (1965) 113.

[53] M. PETERSDORFF and H. SACHS: Uber Spektrum, Automorphismengruppe und Teiler eines Graphen. *Wiss. Z. Techn. Hosch. Ilmenau* 15 (1969) 123-128.

[54] G. POLYA: Kombinatorische anzahlbestimmungen für Gruppen, Graphen und chemische verbindungen. *Acta Math.* 68 (1937) 145-254.

[55] A. PROSKUROWSKI: Search for a unique incidence matrix of a graph. *BIT* 14 (1974) 209-226.

*[56] M. RANDIC: On the recognition of identical graphs representing molecular topology. *J. Chem. Phys.* 60 (1974) 3920-3928.

*[57] R.C. READ: The coding of various kinds of unlabelled trees, in "Graph Theory and Computing", R.C. Read (ed.). Academic Press, N.Y. & Lond. (1972) 153-183.

[58] G. SABIDUSSI: The composition of graphs. *Duke Math. J.* 26 (1959) 693-696.

[59] H. SACHS: Beziehungen zwischen den in einem Graphen enthaltenen Kreisen und seinem charakteristischen Polynom. *Publ. Math. Debrecen* 11 (1964) 119-134.

[60] H. SACHS: Uber Teiler, Faktoren und characterische Polynome von Graphen I. *Wiss. Z. Techn. Hosch. Ilmenau* 12 (1966) 7-12.

[61] H. SACHS: Uber Teiler, Faktoren und characterische Polynome von Graphen II. *Wiss. Z. Techn. Hosch. Ilmenau* 13 (1967) 405-412.

[62] G. SAUCIER: Un algorithme efficace recherchant l'isomorphisme de 2 graphes. *RIRO R-3*, 5 (1971) 39-51.

[63] A.O. SCHWENK: Computing the characteristic polynomial of a graph, in "Graphs and Combinatorics" , *Proc. Capital Conf. on graph theory and combinatorics*. Lecture Notes in Mathematics, Springer-Verlag (1974) 153-162.

[64] W.R. SCOTT: "Group Theory", Prentice-Hall, N.J. (1964).

[65] J.J. SEIDEL: Graphs and two-graphs, in *Proc. 5th S-E Conf. on Combinatorics, Graph Theory and Computing* (1974) 125-143.

[66] S.S. SHRIKHANDE: The uniqueness of the L_2 association scheme. *Ann. Math. Stat.* 30 (1959) 781-798.

*[67] F. SIROVICH: Isomorfismo fra grafi: un algoritmo efficiente per trovare tutti gli isomorfismi. *CALCOLO* 8 (1971) 301-337.

*[68] N. SRIDHARAN: Computer generation of vertex graphs. *Info. Process. Lttrs.* 3 (1974/5) 57-63 and ibid. p.164.

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

[70] E.H. SUSSENGUTH: A graph-theoretic algorithm for matching chemical structures. *J. Chem. Docum.* 5, 1 (1965) 36-43.

[71] R.E. TARJAN: Depth-first search and linear graph algorithms. *SIAM J. Comput.* 1, 2 (1972) 146-160.

[72] G. TINHOFFER: Zur Bestimmung der Automorphismen eines endlichen Graphen. *Computing* 15 (1975) 147-156.

[73] J. TURNER: Generalised matrix functions and the graph isomorphism problem. *SIAM J. Appl. Math.* 16 (1968) 520-526.

[74] J.R. ULLMANN: "Pattern Recognition Techniques". Butterworths, London and Crane Russak, N.Y. (1973).

[75] J.R. ULLMANN: An algorithm for subgraph isomorphism. *JACM* 23, 1 (1976) 31-42.

[76] S.H. UNGER: GIT - A heuristic program for testing pairs of directed line graphs for isomorphism. *CACM* 7, 1 (1964) 26-34.

*[77] L. WEINBERG: A simple and efficient algorithm for determining isomorphisms of planar triply connected graphs. *IEEE Trans. Circuit Theory* 13 (1966) 142-148.

[78] M.B. WELLS: "Elements of combinatorial computing". Pergamon, Oxford (1971).

[79] H. WIELANDT: "Finite Permutation Groups". Academic Press, N.Y. & Lond. (1964).

[80] C.C. YANG: Structural preserving morphisms of finite automata and an application to graph isomorphism. *IEEE Trans. Computers C-24*, 11 (1975) 1133-1139.