Errata in the article by Bosma and Cannon

The article by Bosma and Cannon in the June 1992 issue of CWI Quarterly contains several errors. These errors were detected too late, for which we sincerely apologize.

1. Mathematical errors

Most of these errors occurred because the empty set symbol ($\emptyset$) was not printed.

Page + line	Expression	Must be
130, line 12 from bottom	$N \neq$	$N \neq \emptyset$
130, line 4 from bottom	$\Psi \cap \Psi^g =$	$\Psi \cap \Psi^g = \emptyset$
130, line 3 from bottom	Clearly,	Clearly, $\emptyset$
131, line 18 from bottom	$\neq\neq$	$\neq \emptyset \neq$
138, line 18 from bottom	$\Gamma \neq$	$\Gamma \neq \emptyset$
140, line 1	$B = S =$	$B = S = \emptyset$
140, line 2	$W =$	$W = \emptyset$
144, line 9	$S =,$	$S = \emptyset,$
152, line 3 from bottom	$\frac{1}{log_2(n)} \geq \epsilon$	$\frac{1}{log_2(n)} \geq -\epsilon$

2. E-mail addresses

The e-mail address of Cannon is missing. His e-mail address is

`john@maths.su.oz.au`

The editors

Structural Computation in Finite Permutation Groups

Wieb Bosma* and John Cannon*
School of Mathematics and Statistics,
University of Sydney, Sydney, NSW 2006, Australia.
e-mail: wieb@maths.su.oz.au

1. Introduction

The theory of finite permutation groups has proved to be a particularly fertile field for the design of efficient computer algorithms. Initially, the development of computational methods for permutation groups was driven by the need to construct the various sporadic simple groups that were discovered during the 1960's and 1970's in the course of a project to classify all finite simple groups. More recently, programs implementing such algorithms have found wide application to problems arising both in mathematical research and in areas such as applied combinatorics, cryptanalysis, coding theory and discrete signal processing.

We have available now a rich collection of permutation group algorithms capable of answering a wide range of questions about rather large and complex groups. This area is one of the first branches of 'non-linear' algebra for which a mature and extensive *computational theory* has been developed. Its theoretical importance is, for example, reflected in its application to the computational complexity of combinatorial algorithms. This link is due to the fact that the automorphism group of a combinatorial structure typically has a natural representation as group of permutations acting on the underlying set of the structure. Thus, the automorphism group of a graph is naturally presented as a permutation group on the set of vertices of the graph. In particular, the complexity of the celebrated graph isomorphism problem is equivalent to the complexity of such permutation group problems as that of computing the stabilizer of a set. Thus, using permutation group methods, Luks [31] was able to produce a polynomial-time isomorphism algorithm for the special class of graphs having bounded valence.

The main purpose of this paper is to introduce the reader to practical algorithms for permutation groups of very large degree. The paper will concentrate on the algorithms and theory which enable us to analyze the abstract structure of a permutation group (i.e., determine its composition factors). Our goal is the development of algorithms capable of determining the abstract structure of groups having degrees in excess of a million. The size of the field means that our treatment is, of necessity, highly selective. In particular, we only consider algorithms that are guaranteed to produce correct results. Thus, all algorithms discussed

*This research was supported by the Australian Research Council (Grant A68830699)

will either be deterministic or randomized of the Las Vegas type. Randomized Monte Carlo algorithms for permutation groups are discussed in the paper by Cooperman and Finkelstein appearing in this volume.

Apart from its intrinsic interest, we believe that a careful study of the field will provide important insights for the development of computational theories for other branches of algebra. In the course of our presentation, we will present evidence for two elementary, but important, principles which we believe are generally applicable to the construction of a computational theory for some new branch of algebra:

- The effectiveness of algorithms for computing deep properties of algebraic structures will be critically dependent upon the choice of representation for the *carrier set* of the structure.

- The design of algebraic algorithms is a hierarchical process mirroring the development of the abstract theory. It is possible to design algorithms in a 'layered' fashion so that 'high-level' algorithms intended to probe the deeper properties of an algebraic structure may be constructed out of a standard set of 'fundamental' procedures. The identification of these fundamental procedures and the discovery of fast algorithms to perform them is a key task in the construction of a new computational theory.

In the case of permutation groups, the carrier set (i.e. the set of group elements) is represented by means of a set of transversals over a chain of point stabilizers. This representation is discussed in Section 3, together with methods for its construction. In Sections 4, 5, 6 and 7 we introduce a number of the fundamental procedures for computing with permutation groups. Thus, in Section 4, we present some elementary algorithms which are direct consequences of the machinery set up in Section 3. In Section 5, we show that our representation of a permutation group allows us to design very efficient backtrack searches for various key subgroups. Section 6 introduces the reader to techniques for computing with the standard permutation group homomorphisms and applies these techniques to give a method for finding Sylow p-subgroups. In the final section, we use these basic algorithms as the building blocks for the construction of a rather deep algorithm which finds a composition series for a group.

Since this paper is directed at the reader who may not be familiar with the details of permutation group theory we sketch some of the key ideas. For general background and notation from the theory of permutation groups, the reader is referred to WIELANDT [41]. An elementary exposition of basic permutation group algorithms may be found in the recent book of BUTLER [8]; an earlier survey is given in [15]. The examples quoted in this paper to illustrate the performance of various algorithms were run using the computer algebra system Cayley V3.8.3 [14] on a SUN MP670.

2. Basic Concepts

Finite permutation groups

Let Ω be a finite set having cardinality n. The *symmetric group on* Ω, written S_Ω or S_n, is the group consisting of all bijections of Ω into itself. A subgroup of S_n is called a *permutation group (of degree n)*.

Permutation groups are of interest both in their own right and also because of the fact that any group may be represented as a permutation group. Thus, in the case where G is a finite group of order n, a theorem of Cayley states that G is isomorphic to a subgroup of S_n. More precisely, an isomorphism between any G and a subgroup of S_G is given by the map $g \mapsto \phi_g$, where $\phi_g \in S_G$ acts on the underlying set of G via multiplication by g: $\phi_g(x) = x \cdot g$ for every $x \in G$. This is known as the *right regular representation of* G.

If G is an abstract group, then a *group action* of G on a finite set Ω is a mapping $\Omega \times G \to \Omega$, usually written $x \mapsto x^g$ (satisfying $x^{\mathrm{id}} = x$ and $(x^g)^h = x^{gh}$ for $g, h \in G$). This mapping is obviously a homomorphism $G \to S_\Omega$. Conversely, such a homomorphism corresponds in a natural way to an action of G on Ω: thus, group actions are permutation representations when viewed as maps $G \to S_\Omega$.

A permutation group G will henceforth be described in terms of an action on a finite set Ω, where the homomorphism into S_Ω is usually required to be monomorphic (to have trivial kernel). In other words, we require G to act *faithfully* on Ω, so that the only $g \in G$ leaving every element of Ω invariant will be the identity. Following traditional practice, we shall often refer to the elements of the set Ω as *points*.

Before starting our discussion of permutation group algorithms, we say a little about how particular permutation groups are presented in practice. For simplicity, the n elements of Ω will be identified with the integers $1, 2, \ldots, n$. An element of a permutation group will often be written as a product of disjoint cycles, where a *cycle* $(x_1, x_2, \ldots, x_m)$, represents the permutation $x_i \mapsto x_{i+1}$ (for $1 \leq m - 1$) and $x_m \mapsto x_1$. A finite permutation group G is certainly generated by a finite subset X of G. It is easily proved that at most $n - 1$ generators are needed, but in fact $n/2$ will suffice if $G \ncong S_3$, [34], where n is the degree. The symmetric group S_n, of degree n and order $n!$, is generated by the two cycles $(1, 2)$ and $(1, 2, \ldots, n)$. For the purpose of this paper, we assume that a permutation group G is given in terms of a small generating set X of cardinality k. To avoid repeated calculation of inverses, the generating set X is usually extended to include the inverses of all its members, that is, X is replaced by $X \cup X^{-1}$.

As an aside, we note that given an arbitrary generating set for G it is not easy to find a generating set of minimal size. However, given a set of k generators for G, a simple algorithm may be used to reduce their number to $n(n-1)/2$ in time $O(kn^2)$. There also exists a Monte Carlo algorithm which will produce $O(n)$ generators in time $O(kn \log n)$ (see [3]).

Transitivity and orbits

It is natural to consider the equivalence classes of points of Ω induced by the

action of G, where two points α and β are equivalent if there exists an element $g \in G$ such that $\alpha^g = \beta$. Such an equivalence class is called an *orbit* and is denoted by α^G. A permutation group is said to be *transitive* if it has exactly one orbit, namely Ω; that is, for every pair of points of Ω there is an element of G taking the one to the other. Generalizing this notion, a permutation group is said to be *k-transitive*, $(k \geq 1)$, if for every pair of sequences $(\alpha_1, \ldots, \alpha_k)$, $(\beta_1, \ldots, \beta_k)$, of k distinct points of Ω, there is an element g in G such that $(\alpha_1^g, \ldots, \alpha_k^g) = (\beta_1, \ldots, \beta_k)$.

Restricting the action of G to a single orbit Δ gives rise to a homomorphism of G onto a transitive subgroup of S_Δ. This reduction to a transitive group plays an important role in the design of permutation group algorithms (see section 6).

The orbit Δ of a point α is easily computed by taking images under generators until no new points are found, as in the algorithm below. For many applications it is also desirable to keep a list of elements $x(\beta) \in G$, for $\beta \in \Delta$, such that $\alpha^{x(\beta)} = \beta$. In order to compactly store both the orbit and the elements $x(\beta)$, Sims introduced a linearized tree structure known as a *Schreier vector*. This is a vector v whose entries are indexed by the elements of Ω, such that $v_\alpha = -1$, $v_\beta = 0$ for $\beta \notin \Delta$, and $v_\beta = x$ for $\alpha \neq \beta \in \Delta$, where x is the first generator to produce β as the image of a known point of Δ during the construction of the orbit. Given a point $\beta \in \Delta$ one easily obtains from such a vector a word $w = \prod g_j^{k_j}$ such that $\beta = \alpha^w$.

ALGORITHM ORBIT AND SCHREIER VECTOR.
Input Ω; $\alpha \in \Omega$; a subgroup G of S_Ω generated by $X = \{g_1, g_2, \ldots, g_k\}$.
Output Orbit $O = \alpha^G$ and Schreier vector v.

(0) *Initialize* $O = \{\alpha\}$, $N = \{\alpha\}$, *and the vector* v *by* $v_\alpha = -1$, $v_\beta = 0$ *for* $\alpha \neq \beta \in \Omega$.

(1) *Repeat the following three steps while* $N \neq$:

(i) *Put* $I = \cup_{i=1}^k I_i$, *where* $I_i = N^{g_i}$, *for* $i = 1, \ldots, k$.

(ii) *Put* $N = I \setminus O$, *and for each* $\beta \in N \cap I_i$ *put* $v_\beta = g_i$.

(iii) *Replace* O *by* $O \cup N$.

REMARKS. The construction of an orbit takes time $O(kn)$.

The algorithm is easily adapted to provide a *transitivity test*: calculate the orbit Δ of any point, and if Δ is all of Ω then and only then is G transitive.

Primitivity and blocks
A *block* of G is a subset $\Psi \subset \Omega$ such that for each $g \in G$ either $\Psi \cap \Psi^g =$ or $\Psi \cap \Psi^g = \Psi$. Clearly, , the sets $\{\omega\}$, for ω in Ω, and Ω, are all blocks (the *trivial* blocks of G). A transitive permutation group is said to be *primitive* if these are the only blocks. In a transitive permutation group, the translates Ψ^g of a

block Ψ are also blocks and $\{\Psi^g : g \in G\}$ partitions Ω into equal-sized subsets. The collection of distinct translates of a block Ψ form a *block system*, which is a G-invariant partition of Ω. A transitive group that has a non-trival block is said to be *imprimitive*.

If the transitive group G is imprimitive, then it induces a transitive action on each non-trivial block system. By choosing a maximal block system (with respect to inclusion), we obtain a homomorphism from G onto a primitive group. This is the second main reduction for homomorphism-based permutation group algorithms (see Section 6).

Given points α and β of Ω, the following algorithm constructs a block system such that α and β, lie in the same block Ψ. The system is minimal in the sense that no proper subset of Ψ containing α and β is a block for G.

ALGORITHM BLOCKS
Input Ω; $\alpha, \beta \in \Omega$; a transitive subgroup G of S_Ω generated by $X = \{g_1, g_2, \ldots, g_k\}$.
Output Block system $\mathcal{B}$ containing a block Ψ minimal with respect to the condition that $\alpha, \beta \in \Psi$.

(0) *Create an initial partition of* Ω: $\mathcal{B} = \{\{\alpha, \beta\}, \{\gamma\} : \text{ for each } \gamma \in \Omega \setminus \{\alpha, \beta\}\}$.

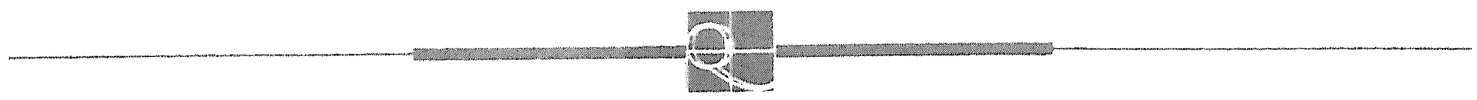
(1) *Repeat the following two steps for* $i = 1, \ldots, k$:

(i) *Calculate* $\mathcal{B}^{g_i} = \{\Psi^{g_i} : \Psi \in \mathcal{B}\}$.

(ii) *Modify* $\mathcal{B}$ *as follows: If* $\Psi' \cap \Psi^{g_i} \neq\neq \Psi^{g_i} \cap \Psi''$ *for two sets* $\Psi', \Psi'' \in \mathcal{B}$ *and some* i, *then replace* Ψ' *and* Ψ'' *by* $\Psi' \cup \Psi''$.

REMARKS. It is interesting to note that this algorithm can be made to run in time $O(kn \log n)$ by a careful choice of data structures. For example, Atkinson [1] uses a time-varying integer-valued function on the points of Ω that agrees on two points if and only if they are currently known to lie in the same block.

By applying the Blocks Algorithm to each pair α, β of distinct points chosen from Ω, one can determine whether G is *primitive* in time $O(kn^2 \log(n))$. However, for large degrees this approach is much too inefficient and so, in practice, the following reduction is employed. Let G_α be the subgroup of G that fixes the point α (see also the next section), and let $\alpha_1, \ldots, \alpha_r$ be representatives for the orbits of G_α on Ω. If none of the pairs (α, α_i) for $i = 1, \ldots, r$ lies in a proper block for G, then G is primitive. In order to take advantage of this reduction we need generators for the stabilizer G_α which may be very expensive to compute if it is not already known. In this situation, we employ the random Schreier-Sims method (see section 3) to quickly compute some subgroup H of G_α and we then apply then the Blocks Algorithm to the pairs (α, γ_i), for $i = 1, \ldots, s$, where $\gamma_1, \ldots, \gamma_s$ are representatives for the orbits of H.



EXAMPLES:
(1) The wreath product (with product action) of the projective group PGL(2, 7) and the projective group PGL(2, 5), is a primitive group of degree 262, 144. A straightforward implementation of the above ideas establishes primitivity in 550 seconds. Of this time, approximately 500 seconds was spent in constructing an approximation H of a one-point stabilizer using twelve pseudo-random elements. The subgroup H has 7 orbits and each application of the Blocks Algorithm took, on average, 5 seconds.

(2) The wreath product (with product action) of the projective group PGL(2, 9) and the symmetric group S_6, is a primitive group of degree $1,000,000$. The execution time required to prove the primitivity of this group was $2,350$ seconds. Approximately $2,270$ seconds was spent in constructing an approximation H to G_α having 7 orbits on Ω. Each application of the Blocks Algorithm took, on average, 14 seconds.

Regularity
There is a strong relation between most of the notions introduced so far and point stabilizers: the *point stabilizer* G_β of a point $\beta \in \Omega$ is the subgroup of G fixing β. It is clear that the image of some $\beta \in \Omega$ under two elements $g_1, g_2 \in G$ is the same if and only if $g_1 g_2^{-1}$ fixes β, that is

$$\beta^{g_1} = \beta^{g_2} \quad \Longleftrightarrow \quad g_1 g_2^{-1} \in G_\beta.$$

The first implication of this is that the length of the orbit of β must equal the number of distinct (right) cosets $[G : G_\beta]$ of its point stabilizer. In fact the Schreier vector is an encoding of a set of coset representatives for G_β in G.

It is not hard to show that a transitive permutation group has no non-trivial blocks (is primitive) if and only if the subgroups G_β are maximal in G. For, let N be a subgroup containing G_β. Then the orbit $\Psi = \beta^N$ must be a block: suppose there is some point $\alpha \in \Psi \cap \Psi^g$; then $\alpha = \beta^h = \beta^{h'g}$ for some $h, h' \in N$, hence $h'gh^{-1}$ fixes β and $g \in G_\beta$. But then $\Psi^g = \Psi$. The converse is equally easy.

If, at the other extreme, $G_\beta = \{\mathrm{id}\}$ for every point, then G is called *semiregular*. Note that all orbits in a semiregular group must have length $[G : G_\beta] = |G|$; therefore a transitive group is semiregular (and then called *regular*) if and only if its order equals its degree.

ALGORITHM REGULARITY
Input Ω; a transitive permutation group G generated by $X = \{g_1, g_2, \ldots, g_k\}$.
Output Returns true if G is regular, false otherwise.

(0) *Initialize* f *to be* true. *Choose a point* $\alpha \in \Omega$.

(1) *Repeat the following three steps for* $i = 1, \ldots, k$:

(i) *Calculate* $\gamma = \alpha^{g_i}$ *and define the map* $z : \Omega \to \Omega$ *by* $z(\omega) = \gamma^w$, *where* $w \in G$ *is such that* $\omega = \alpha^w$, *for each point* $\omega \in \Omega$.

(ii) *If z is not bijective, set f equal to* false *and go to Step* (2).

(iii) *If z does not commute with g_j, for some j in $1, 2, \ldots, k$, then set f to* false *and go to Step* (2).

(2) *Return f.*

REMARKS. This algorithm (due to Sims, unpublished) checks that the centralizer of G in S_Ω acts transitively. Then for any point β there exists an element z in that centralizer such that $\alpha^z = \beta$. Therefore $G_\alpha = z^{-1}G_\alpha z = G_{\alpha^z} = G_\beta$, for every β, and since G is transitive, $G_\alpha = \{\mathrm{id}\}$.

To conclude this section we mention Frobenius groups, which will briefly appear in Section 7. A *Frobenius group* is a non-regular transitive permutation group in which every element $g \neq \mathrm{id}$ fixes at most one point. (The elements having no fixed points form, together with id, the *Frobenius kernel.*) A Frobenius group is not regular, but $G_{\alpha,\beta} = \{\mathrm{id}\}$ for all points $\alpha \neq \beta$.

3. STABILIZERS, BASES AND STRONG GENERATORS

Stabilizers

A *transversal* T for a subgroup H of a group G is a complete set of (right) coset representatives for H in G (we assume that T contains id).

SCHREIER'S LEMMA. *Let H be a subgroup of $G = \langle g_1, \ldots, g_k \rangle$. Then H is generated by the elements $h_{i,j} = t_i g_j \phi(t_i g_j)^{-1}$, where t_i runs through a transversal T for H in G, and ϕ maps an element of g to the representative in T for the coset Hg.*

As a consequence, it is possible to determine generators for the point stabilizer G_α of a point α in time $O(kn^2)$: simply take all Schreier generators $g_\omega g_j g_{\omega^{g_j}}^{-1}$, where the g_j generate G, and, for any ω in the orbit α^G, the element g_ω (obtained from the Schreier vector) is such that $\alpha^{g_\omega} = \omega$.

Furthermore, it is now possible to give an algorithm to answer one of the most basic questions that one might ask about a given permutation group: what is its order? Since the order of G is the product of the length of the orbit α^G and the order of the point stabilizer G_α, one can determine the order by applying the Orbit algorithm and the above point stabilizer construction repeatedly to a succession of points $\alpha_1, \alpha_2, \ldots, \alpha_r$ until $G_{\alpha_1,\alpha_2,\ldots,\alpha_r} = \{\mathrm{id}\}$.

However, if the subgroup H has index r in G, then Schreier's Lemma yields $1 + r(k-1)$ generators for H. For a finite group G, the subgroup H will usually be generated by a tiny fraction of this number. Consequently, it is impractical to use the above point stabilizer construction iteratively, to obtain the order, since the number of Schreier generators constructed as one moves down the subgroup chain rapidly becomes unmanageable. Sims therefore introduced a slightly different strategy in order to avoid constructing redundant generators.

Before describing this strategy, we introduce the fundamental notions of a base and strong generating set for a permutation group.

Base and strong generators
In order to design efficient algorithms for permutation groups it is necessary to develop a computationally efficient method of representing the *set* of elements of the group. In many branches of algebra, structures can be presented in terms of some ascending chain of substructures, where the substructures are described by means of a basis of some kind. Thus, a vector space may be presented by an ascending chain of subspaces, the i-th member of which is spanned by the first i elements of a basis (for $i = 0, 1, \ldots$, up to the dimension). Although groups do not possess the linear structure that leads to the usual concept of a basis, Sims [37] introduced a chain of subgroups which plays a very similar role to that played by the chain of subspaces defining the echelon form representation of a subspace of a vector space.

A *base* B for a permutation group G is a sequence of distinct points $B = (\beta_1, \ldots, \beta_k)$ from Ω such that the only element of G that fixes each of them is the identity. In other words the pointwise stabilizer $G_{\beta_1,\ldots,\beta_k}$ is trivial. We consider the chain of subgroups of G consisting of the pointwise stabilizers $G^{(i)} = G_{\beta_1,\ldots,\beta_{i-1}}$ of the first $i-1$ base points, for $i = 1, 2, \ldots, k+1$. Then:

$$\{\mathrm{id}\} = G^{(k+1)} \leq G^{(k)} \leq \cdots \leq G^{(2)} \leq G^{(1)} = G.$$

A *strong generating set* S *for* G *relative to* B is a set of generators for G which includes generators for each stabilizer $G^{(i)}$ in the chain above, so for $i = 1, 2, \ldots, k+1$,

$$G^{(i)} = \langle G^{(i)} \cap S \rangle 13847.$$

A transversal for $G^{(i+1)}$ in $G^{(i)}$ will be denoted by $U^{(i)}$. The sequence of transversals $U^{(1)}, \cdots, U^{(k)}$ provides us with the required representation of the set G since every element $g \in G$ has a unique representation $g = u_k u_{k-1} \cdots u_1$ with $u_i \in U^{(i)}$. Furthermore, $\Delta^{(i)} = \beta_i^{G^{(i)}}$, is called the i-th *basic orbit*. Note that the order of the group can be read off from the transversals, since $|G| = |U^{(k)}| \cdot |U^{(k-1)}| \cdots |U^{(1)}|$, and that transversals can be obtained in the form of the Schreier vectors together with the orbits, using the Orbit Algorithm. Finally, if $B = (\beta_1, \ldots, \beta_k)$ is a base, and $g \in G$, then $B^g = (\beta_1^g, \ldots, \beta_k^g)$ is called the *base image* of g (relative to B). Note that its base image uniquely determines g. For brevity, we will often write *BSGS* for *base and strong generating set*.

One of the important advantages of representing elements by means of base images arises from the fact that for many interesting groups, the size of a base B may be rather small compared to the degree of the group. For example, linear groups, regarded as permutation groups acting on a vector space, always have a base of size at most $\log n$. The worst case occurs for the symmetric group S_n: it is clear that a base must contain $n-1$ points (and that any $n-1$ will do); choosing $B = (n, n-1, \ldots, 2)$ one finds that $G^{(i)} \cong S_{n-i+1}$, the full symmetric group acting on the first $n-i+1$ points (which form the i-th basic orbit). Then

$S = \{(1,2),(2,3),\ldots,(n-1,n)\}$ is easily seen to be a strong generating set for S_n, with respect to B.

Before we describe a method for constructing a BSGS, we show how knowledge of it provides us with a membership test.

ALGORITHM STRIP

Input Ω; a permutation $g \in S_\Omega$; a subgroup G of S_Ω, together with base B and strong generating set S, basic orbits $\Delta^{(i)}$ and transversals $U^{(i)}$.

Output An element $h \in S_\Omega$ and elements $u_j \in U^{(j)}$ for $j = 1,2,\ldots,k$ defined as follows: if $g \in G$ then $h = \mathrm{id}$ and $g = u_k u_{k-1} \cdots u_1$; if $g \notin G$ then $h \neq \mathrm{id}$.

(0) *Initialize* $h = g$, $i = 1$, $u_j = \mathrm{id}$ *for* $j = 1,2,\ldots,k$, *and* $f = true$.

(1) *Repeat the following step until either* $i = k+1$ *(in which case* $g \in G$*) or* $f = \mathrm{false}$ *(in which case* $g \notin G$*).*

(i) *Calculate* β_i^h*; if* $\beta_i^h \notin \Delta^{(i)}$ *then* $f = \mathrm{false}$*, else let* u_i *be the unique element of* $U^{(i)}$ *for which* $\beta_i^h = \beta_i^{u_i}$*, replace* h *by* hu_i^{-1} *and replace* i *by* $i+1$.

REMARKS. Algorithm Strip may be used as a *membership test* for G.

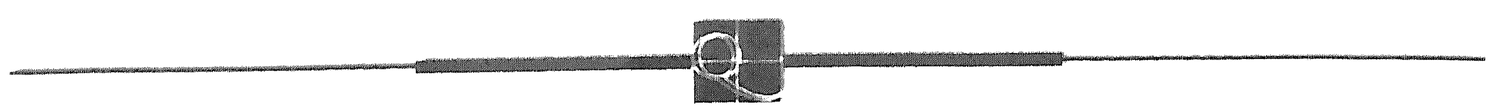
In fact Algorithm Strip may be used in the slightly more general situation where a base and strong generating set are not yet known for the whole of G, but only for some subgroup of G. For this application, Strip is modified to return a *drop-out level* j in case g does not lie in the subgroup of G defined by the transversals $U^{(1)},\cdots,U^{(k)}$. The drop-out level j is defined to be the smallest integer such that $\beta_j^g \notin \Delta^{(j)}$. The permutation h returned by Strip will be referred to as the *residue* of g.

We now present a practical algorithm for determining a base and strong generating set for a permutation group. Again the main tool is Schreier's Lemma but this time, rather than building up a complete point stabilizer as suggested before, whenever a new Schreier generator x is constructed, we use the modified form of Strip to test whether x lies in the subgroup H of G defined by the existing partial BSGS. If it does not lie in H, we use it to create new Schreier generators at the drop-out level for x and if it does lie in H we simply discard it.

ALGORITHM SCHREIER-SIMS

Input Ω; a subgroup $G = < g_1,\ldots,g_m >$ of S_Ω.

Output A base B and a strong generating set S relative to B for G.



(0) *Initialize* $B = (\beta_1)$ *(any point of* Ω *moved by some generator* g_i *of* G*),*
$S = \{g_1, \ldots, g_m\}$, $S^{(1)} = S$, $\Delta^{(1)} = \beta_1^{<S^{(1)}>} = \beta_1^G$ *and* $i = 1$.
While $i > 0$ *repeat Steps* (1) *and* (2).

(1) *For each* $\delta \in \Delta^{(i)}$ *and each* $s \in S^{(i)}$ *perform Steps* (i)–(iii).

(i) *Let* g_δ *be an element of* G *such that* $\beta_i^{g_\delta} = \delta$ *and define* $g = g_\delta s g_{\delta^s}$.

(ii) *Apply algorithm Strip to* g *and* $< S^{(i+1)} >$*; if* $g \in < S^{(i+1)} >$ *continue with Step* (1). *If* $g \notin < S^{(i+1)} >$ *let* h *be the residue of* g *and* j *its drop-out level, replace* S *by* $S \cup \{h\}$*; if* $j = k + 1$ *then replace* B *by* $(\beta_1, \beta_2, \ldots, \beta_k, \beta_{k+1})$*, where* β_{k+1} *is a point moved by* h *and replace* k *by* $k + 1$.

(iii) *Recalculate* $S^{(l)} = \{s \in S : \beta_1^s = \beta_1, \ldots, \beta_{l-1}^s = \beta_{l-1}\}$, $\Delta^{(l)} = \beta_l^{<S^{(l)}>}$*, and the Schreier vectors for* $l = i+1, i+2, \ldots, j$*, and replace* i *by* j.

(2) *If* S *has not been modified in Step* (1)*, replace* i *by* $i - 1$.

REMARKS. The theoretical complexity of BSGS algorithms is heavily dependent upon the choice of data structures used to store the transversals $U^{(i)}$. Sims stored each transversal $U^{(i)}$ in the form of a *Schreier vector* and the running time of a variation of his original algorithm was bounded by $O(n^6 + kn^2)$ [20]. Using the so-called *labelled branching* data structure for the $U^{(i)}$, JERRUM [23] described a variant of the original Sims algorithm with running time $O(n^5 + kn^2)$. KNUTH [27] describes another variant with running time $O(n^5 + kn^2)$.

The random Schreier method and verification

For larger degrees the Schreier-Sims algorithm considers too many Schreier generators in Step (1) and so a different approach must be adopted. First, a "probable" BSGS for G is constructed, and then an algorithm is applied which either *verifies* that the BSGS is correct, or establishes that it is incomplete.

A "probable" BSGS may be constructed very quickly by using a fixed number of randomly chosen elements of G in place of the products $t_i g_j$ in Schreier's lemma (*Random Schreier-Sims algorithm*). Thus, instead of taking every pair δ and s to construct generators g in Step (1)(i), one chooses a random element $g \in G$, and keeps repeating this until for m consecutive choices of g nothing happens in Step (1)(ii) (that is, each g lies in the subgroup of G found so far); here m is an integer that is chosen beforehand to determine the probability that the algorithm returns a complete base and strong generating set. An important practical issue that arises with the Random Schreier-Sims method concerns the need to somehow choose random elements $g \in G$. This is usually done by taking random *words* in the generators g_i; to make these reasonably random, the words have to be very long (depending on the degree of the group). It is easily shown that for truly random g the probability that g lies in the subgroup of G defined by an incomplete base and strong generating set is less than $1/2$.

The main inductive step in BSGS verification is the following: Suppose H is a subgroup of $G^{(2)}$ with a certified BSGS. If we can show that $H = G^{(2)}$, then we will have verified the correctness of the BSGS for G. In practice, H will be the approximation to $G^{(2)}$ constructed by the Random Schreier-Sims algorithm. The first verification algorithm was developed by Sims and published by LEON [28] and involves using the Todd-Coxeter algorithm [40, 21] to construct a presentation for G in terms of strong generators. A presentation on the strong generators for H is assumed to be known by induction. The verification involves comparing the index of H in G with the length of the orbit β_1^G. This method made it practical to construct BSGSs for groups having degree in the thousands. In 1986, Brownie and Cannon developed a new verification algorithm based on an idea of Sims. Instead of testing each of the Schreier generators given in the above lemma for membership of H, we test a much smaller subset defined in terms of representatives for the orbits of certain one-point and two-point stabilizers. This algorithm, as implemented in Cayley, has constructed and verified the correctness of BSGSs for some groups having degrees in excess of a million.

EXAMPLES:
(1) The Fischer sporadic simple group Fi'_{24} has order 1255205709190661721292800 and possesses a permutation representation of degree 309,936. The random Schreier-Sims algorithm constructs a BSGS for the group in 4176 seconds. A further 3930 seconds is required by the Brownie-Cannon-Sims algorithm to verify the correctness of this BSGS.

(2) The Harada-Norton sporadic simple group has order 273030912000000 and possesses a permutation representation of degree 1,140,000. The random Schreier-Sims algorithm constructs a BSGS in 8000 seconds. A further 19,700 seconds is required to verify the correctness of this BSGS.

COOPERMAN and FINKELSTEIN [16] describe an algorithm which verifies strong generation in $O(n^4)$ time. Experimental work is needed in order to establish whether or not the Cooperman-Finkelstein algorithm is a practical competitor to the Brownie-Cannon-Sims algorithm.

Base change
In permutation group computation, it frequently turns out to be desirable to compute a strong generating set for the group G relative to a base different from the current one of G. Of course, we could achieve this by running the Schreier-Sims algorithm again using the new base to determine the choice of stabilizers. However, once we have any BSGS for G there are much faster ways of computing a strong generating set relative to a new base. This technique will be referred to as *base change*.

Suppose $B = (\beta_1, \ldots, \beta_k)$ is a base and S is a strong generating set for G relative to B. Clearly, if $\gamma \notin B$, then $B' = (\beta_1, \ldots, \beta_k, \gamma)$ is also a base for G, and

S is a strong generating set also relative to B'. Similarly, if $B'' = (\beta_1, \ldots, \beta_{k-1})$ is a base for G then again S is a strong generating set relative to B''. Finally, any permutation of B will also form a base (but usually S will no longer constitute a strong generating set relative to such a base). Any such permutation of the base B may be obtained by performing a sequence of *adjacent transpositions*, that is, transpositions of adjacent pairs of points β_j and β_{j+1}. Algorithm Interchange below constructs a strong generating set for the base B as modified by such a transposition. Given this algorithm, changing base from B to $C = (\gamma_1, \ldots, \gamma_l)$ proceeds as follows: First replace B by $B' = (\beta_1, \ldots, \beta_k, \gamma_1, \ldots, \gamma_l)$. Now using Algorithm Interchange repeatedly, find a strong generating set S' for $B'' = (\gamma_1, \ldots, \gamma_l, \beta_1, \ldots, \beta_k)$ and return $B''' = (\gamma_1, \ldots, \gamma_l)$ and S'.

ALGORITHM INTERCHANGE

Input A base $B = (\beta_1, \ldots, \beta_k)$ and strong generating set S relative to B for a permutation group G, an integer j with $1 < j \leq k$, and basic orbits $\Delta^{(j)}, \Delta^{(j+1)}$.

Output A strong generating set for G relative to $B' = (\beta_1, \ldots, \beta_{j-1}, \beta_{j+1}, \beta_j, \ldots, \beta_k)$.

(0) *Initialize* $T = S^{(j+2)}$, $\Gamma = \Delta^{(j)} \setminus \{\beta_j, \beta_{j+1}\}$, *and* $\Delta = \beta_j^{<T>} = \{\beta_j\}$.

(1) *While* $\Gamma \neq$ *repeat Steps* (i) *and* (ii).

(i) *Choose* $\gamma \in \Gamma$ *and let* $x \in G^{(j)}$ *be such that* $\gamma = \beta_j^x$.

(ii) *If* $\beta_{j+1}^{x^{-1}} \notin \Delta^{(j+1)}$, *then replace* Γ *by* $\Gamma \setminus \gamma^{<T>}$; *if* $\beta_{j+1}^{x^{-1}} \in \Delta^{(j+1)}$ *then let* $y \in G^{(j+1)}$ *be such that* $\beta_{j+1}^y = \beta_{j+1}^{x^{-1}}$, *replace* T *by* $T \cup \{xy\}$, *recalculate* $\Delta = \beta_j^{<T>}$ *and replace* Γ *by* $\Gamma \setminus \Delta$.

(2) *Return* $S' = S \cup T$.

REMARKS. The new set S' is obtained by adding extra generators T for the new $G^{(j+1)}$, which is the only stabilizer that changes. In fact, the new stabilizer $G^{(j+1)}$ equals $G^{(j)}_{\beta_{j+1}}$, so is contained in the old $G^{(j)}$, and therefore the new $(j+1)$-th basic orbit is contained in the old j-th basic orbit $\Delta^{(j)}$. It follows that all points in the new $(j+1)$-th basic orbit must be contained in the initial set Γ, and at every stage either no point of $\gamma^{<T>}$ lies in the new $(j+1)$-th basic orbit, or all of them do. Now a point β_j^x is in the $(j+1)$-th basic orbit of B' if and only if $\beta_{j+1}^{x^{-1}}$ is in the $(j+1)$-th basic orbit of B.

This algorithm, which is due to SIMS [38, 39], runs in time $O(n^3)$, so that a complete base change as outlined above will take $O(kln^3)$. Some modifications which speed up this procedure are described in [8]. In [5] a variant with running time $O(kn^3)$ is described.

A quite different approach involves using the random Schreier-Sims algorithm to construct a set of strong generators relative to the new base. In this situation, since the order of the group is already known, the verification of correctness of the BSGS is trivial. This approach was first discussed by LEON [28] and is employed in the Brownie-Cannon-Sims verification algorithm. Recently, COOPERMAN, FINKELSTEIN and SARAWAGI [18] have described a random base change algorithm having a very favourable running time.

4. SOME APPLICATIONS

In this section we mention some easy applications of the algorithms in the previous section to the construction of various interesting subgroups of a given permutation group.

The first application is (an alternative way) to compute the stabilizer $G_{\alpha_1,\ldots,\alpha_l}$ for any sequence of distinct points $\alpha_1, \ldots, \alpha_l$.

ALGORITHM SEQUENCE STABILIZER

Input Ω; a subgroup G of S_Ω generated by $\{g_1, \ldots, g_k\}$ and equipped with a base $B = \{\beta_1, \ldots, \beta_m\}$ and strong generating set; a sequence $A = (\alpha_1, \ldots, \alpha_l)$ of points of Ω.

Output $G_{\alpha_1,\ldots,\alpha_l}$

(1) *Apply the Interchange Algorithm repeatedly to change the base to* $(\alpha_1, \ldots, \alpha_l, \beta_{l+1}, \ldots, \beta_r)$, *where* $\beta_{l+1}, \ldots, \beta_r \in B$.

(2) *Return the* $(l+1)$*-th stabilizer* $G^{(l+1)}$.

REMARKS. It should perhaps be remarked here that it is much harder to determine the *set stabilizer* $G_{\{\alpha_1,\ldots,\alpha_l\}}$, that is, the subgroup whose elements leave $\{\alpha_1, \ldots, \alpha_l\}$ invariant as a set but not necessarily pointwise. In fact the problem of finding set stabilizers has been shown to be at least as hard as the well-known problem of testing graph isomorphism (which is one of the famous problems in complexity theory that is not known to be solvable in polynomial time nor known to be NP-complete). See also sections 5 and 6.

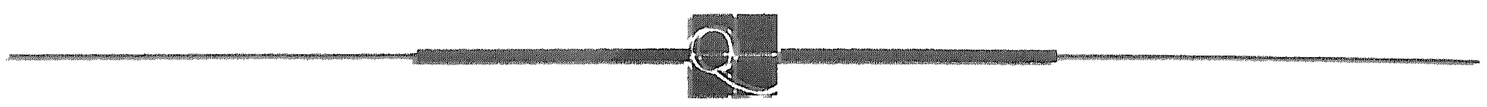
The second application is that of determining the normal closure of a given subgroup (or subset) of G. The *normal closure* $\mathrm{ncl}_G(Y)$ of a subset Y in G is the intersection of all normal subgroups of G containing Y.

It is readily proved that $\mathrm{ncl}_G(Y)$ is generated by elements $g^{-1}yg$, for $g \in G$ and $y \in Y$.

ALGORITHM NORMAL CLOSURE

Input Ω; a subgroup G of S_Ω generated by $\{g_1, \ldots, g_k\}$; a subset $Y \subset G$.

Output Base and strong generators for $\mathrm{ncl}_G(Y)$



(0) *Initialize* $N = Y$ *and* $W = Y$ *and* $B = S =$.

(1) *Repeat the following step until* $W =$:

(i) *Choose* $w \in W$, *replace* W *by* $W \setminus w$ *and perform Step* (a) *for every generator* g_i *of* G.

(a) *Let* $h = x^{-1}wx$; *apply Algorithm Strip to test whether or not* $h \in N$; *if* $h \notin N$, *then replace* N *by* $N \cup \{h\}$, *use a version of the Schreier-Sims Algorithm to replace* B *by a base and* S *by a strong generating set for* $< N >$, *and replace* W *by* $W \cup \{h\}$.

(2) *Return the base* B *and strong generating set* S *for* $< N >= \mathrm{ncl}_G(Y)$.

REMARKS. In [17] an algorithm is described that uses only one application of the Schreier-Sims Algorithm, albeit in a group of degree $2n$.

We now have available the tools to calculate commutator subgroups. If H and K are subgroups of G, the *commutator subgroup* $[H, K]$ is the subgroup of G generated by the commutators $[h, k] = h^{-1}k^{-1}hk$, with $h \in H, k \in K$. If generators for normal subgroups H and K are given, $H =< h_1, \ldots, h_r >$ and $K =< k_1, \ldots, k_s >$ say, then $[H, K] = \mathrm{ncl}_G(\{[h, k] : h \in \{h_1, \ldots, h_r\}, k \in \{k_1, \ldots, k_s\}\})$. So commutator groups of normal subgroups may be found using the Normal Closure Algorithm above. This immediately enables us to calculate the derived series and the lower central series for a permutation group.

The *derived series* for G is the chain of subgroups

$$G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(j)} \geq G^{(j+1)} \geq \cdots,$$

where $G^{(j+1)} = [G^{(j)}, G^{(j)}]$. If the series stabilizes at $G^{(j)}$, that is, if $G^{(k)} = G^{(j)}$ for $k \geq j$, then $G^{(j)}$ is called the *solvable residual* of G, and is written $G^{(\infty)}$. A group is *solvable* if $G^{(\infty)} = \mathrm{id}$. Hence we have a *solvability test*.

The *lower central series* for G is the chain of subgroups

$$G = \gamma_1(G) \geq \gamma_2(G) \geq \cdots \geq \gamma_j(G) \geq \gamma_{j+1}(G) \geq \cdots,$$

where $\gamma_{j+1}(G) = [\gamma_j(G), G]$. If $\gamma_l(G) = \mathrm{id}$ for some l, then G is *nilpotent*. Our ability to compute commutator subgroups provides a *nilpotency test* for permutation groups.

5. Backtrack Search Algorithms

Backtrack search

An important technique in permutation group theory is that of *backtrack search*. It is typically applied to find the subgroup of a given permutation group whose elements satisfy some (group theoretic) property, such as centralizing a given element.

Backtrack search is a standard technique in computational combinatorics being used, for example, in the construction of automorphism groups of combinatorial structures, such as graphs, codes, block designs etc. Typically, these automorphism groups are regarded as subgroups of the full symmetric group of the underlying point set.

Although the principles of backtrack search are not hard to understand, they are more easily explained in the context of particular examples than in general terms. Therefore, we will illustrate backtrack search using the (simplistic) example of finding the automorphism group of the cube graph (in three dimensions).

EXAMPLE. Consider the automorphism group of the cube graph. Obviously, this will be a subgroup of the full symmetric group on 8 points S_8. The group G we are seeking will act on the set Ω consisting of the 8 vertices of the graph, as in Figure 1.

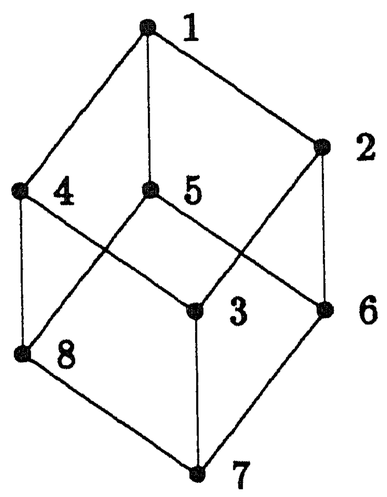


FIGURE 1.

The elements of the group S_8 may be represented in the form of a *tree* having 8! leaves and just one root: Start at the root, and let one branch correspond to every possible image of the point 1 and label each node with that image (these are the 8 nodes at level 1). Continue this process for every new node, until all possibilities are exhausted (at level k there will be $n+1-k$ branches originating from each node).

This tree is merely a way of enumerating all elements of S_8, so that each element corresponds to a path from the root to some leaf. In fact we could have

omitted the leaves of the tree, in which case we would have obtained a tree of height 7, where each path from root to leaf would correspond to an image of the base $(1, 2, 3, 4, 5, 6, 7)$.

Backtrack search traverses the search tree in such a way that large subtrees may be ignored. Thus, the automorphism group G of the cube graph may be found by searching the tree just constructed. The first major improvement is to restrict to a base for G rather than to a base for S_8. In this case it is easy to see that the cube is fixed pointwise if we fix three suitable points. Choosing the points $1, 2, 3$ for our base (it is usual to rearrange points in such a way that the base points come first), this means that we have reduced the tree search to the first three levels of the tree for S_8, part of which is shown in Figure 2.

Given a base, the next thing we require is an algorithm that, given a base image $(\gamma_1, \gamma_2, \ldots, \gamma_k)$ as input, either extends this image to a unique automorphism in G, or reports that no such extension exists. For the cube graph this is very easy to do, using just the adjacency relation. For example, it is clear from Figure 1 that the base image $(1, 4, 3)$ uniquely extends to the automorphism interchanging the points 6 and 8, and the points 2 and 4, while leaving the others invariant. On the other hand, the base image $(1, 3, 2)$ cannot be extended to an automorphism, since the image 3 of 2 is no longer adjacent to the image 1 of 1.

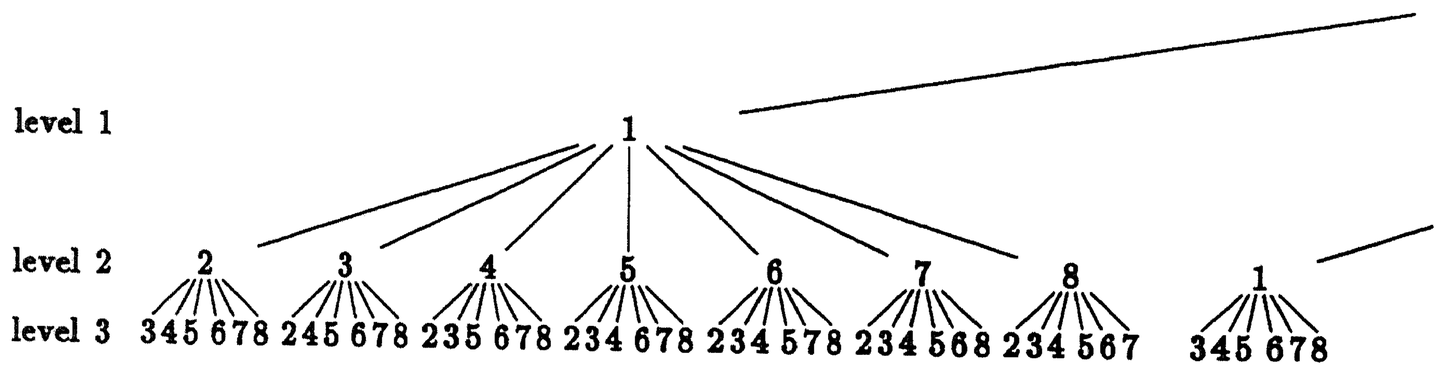


FIGURE 2.

Proceeding in this fashion, it is easy to enumerate all 48 elements of G. But this is only practical because our example is so small. We seek a more efficient way of organizing the search which, moreover, provides more structural information than just the list of elements for G. This is where the backtrack search starts to pay off: firstly, it will actually consider far fewer than $8 \cdot 7 \cdot 6$ elements, and secondly, it will yield *strong generators* for G with respect to the chosen basis.

We will search the tree using the ordering on the base images induced by the usual ordering according to magnitude of the points: that is, we use the lexicographical ordering on the base images. Thus $(2,3,4) < (3,1,2) < (3,1,4)$ etc. Note that, since base images correspond injectively to group elements, this gives an ordering on the elements of G. In Figure 2 this means that we traverse the tree from left to right.

We start the backtrack search at level 3, looking at all base images of the form $(1,2,*)$, i.e. we search the subgroup $G^{(3)} = G_{1,2}$.

The first element to be considered is the base image $(1,2,3)$, corresponding to the identity element of G. This is, of course, an element of G. The next two base images, $(1,2,4)$ and $(1,2,5)$ do not extend to automorphisms, since neither 4 nor 5 is adjacent to 2 in the graph. But $(1,2,6)$ does extend to an automorphism, namely, in cycle notation (leaving out cycles of length 1), to $g_1 = (3,6)(4,5)$, which will be the first generator of G. Since neither $(1,2,7)$ nor $(1,2,8)$ extend to automorphisms, g_1 will be the only generator for the subgroup $G_{1,2}$ that fixes the first two base points. The search has exhausted all possibilities at level 3, so we go up one level and try the next possibility for the image of the second base point, that is, all base images starting with $(1,3)$: we are now searching $G_1 \setminus G_{1,2}$ for elements of G. However, no such element will extend to an automorphism, and we go on to base images starting with $(1,4)$. In fact, we may use the 'extend' algorithm to determine, given an initial segment of a base image of length 2, either the (two) possible extensions or the fact that it cannot be extended. For $(1,4)$ one obtains the two possible extensions $(1,4,3)$ and $(1,4,8)$. The first of these corresponds to the second generator $g_2 = (3,4)(6,8)$ of G. The base image $(1,4,8)$ corresponds to the element $(2,4,5)(3,8,6) = g_1 g_2$ of G, which is already in the subgroup $< g_1, g_2 >$ of G. If we continue in this way, we still enumerate all elements of G, perhaps recognizing now when we do not have to extend our set of generators.

However, there is a technique for pruning the search tree using the subgroup generated so far. This is called the *first in the orbit test.* The orbits of $< g_1, g_2 >$ are $\{1\}, \{2,4,5\}, \{3,6,8\}, \{7\}$. It is only necessary to consider new base images for which the image is first in its orbit (that is, if it is the smallest element in the orbit). In general, when searching $G^{(i)} \setminus G^{(i+1)}$, one only need consider base images $(\gamma_1, \ldots, \gamma_{i-1}, \gamma_i, \ldots, \gamma_k)$ with γ_i first in its orbit. In our example, we are currently searching $G^{(2)} \setminus G^{(3)} = G_1 \setminus G_{1,2}$, so we need only consider the base images $(1,2,*)$ and $(1,3,*)$ at this point: both have already been considered in fact, hence we go up one level and start searching $G^{(1)} \setminus G^{(2)} = G \setminus G_1$. Here again we only need consider base images of the form $(2,*,*)$ and $(3,*,*)$. The first partial image in G_2 that extends, $(2,1,*)$, yields $(2,1,4)$ (and $(2,1,5)$). Now $(2,1,4)$ corresponds to the element $(1,2)(3,4)(5,6),(7,8) = g_3$ of G, which is not in $< g_1, g_2 >$. But the only orbit of $< g_1, g_2, g_3 >$ is in fact $\{1,2,3,4,5,6,7,8\}$, and since now only 1 is first in the orbit and we have already dealt with that case, we may go up one level. Since we are already at the top level, we terminate having found the complete automorphism group, $G =< g_1, g_2, g_3 >$.

Figure 3 shows the small part of the tree which has been searched.

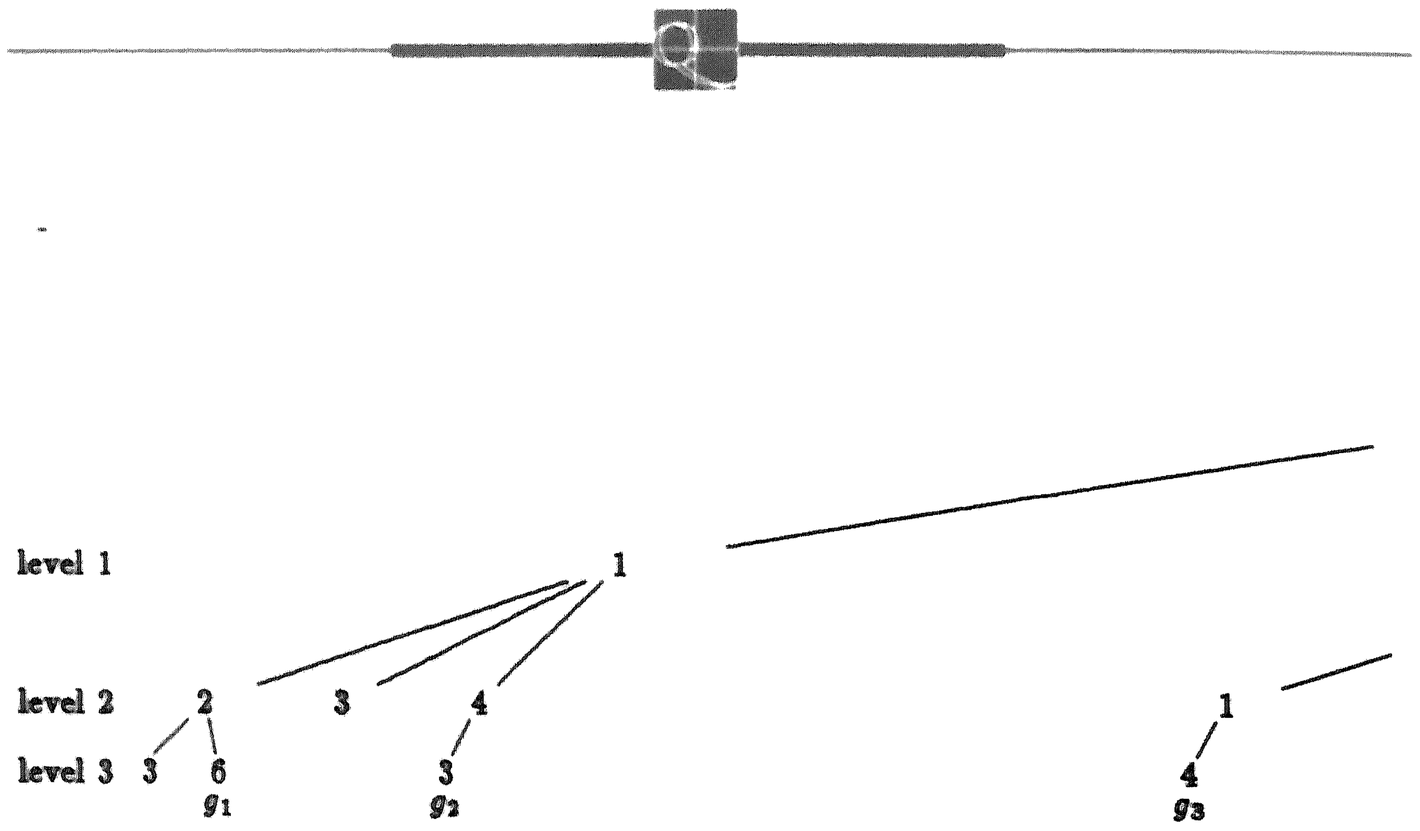


FIGURE 3.

A rudimentary form of the backtrack algorithm used to generate the group G is as follows.

ALGORITHM BACKTRACK

Input Ω; a subgroup H of S_Ω equipped with base and strong generating set; a subgroup G of H with base $(\alpha_1, \ldots, \alpha_k)$; an algorithm **extend** that decides whether a given base image extends to an element of G, and if so, returns that element.

Output Strong generators S for G.

(0) *Initialize* $S = \langle\rangle$, $l = k$, *the one element orbits* $\{1\}$, $\{2\}$, ... , $\{|\Omega|\}$, *and* $\beta = (\beta^{(1)}, \beta^{(2)}, \ldots, \beta^{(k)}) = (1, 2, \ldots, k-1, k-1)$.

Repeat the following step until $l = 0$.

(1) *Using the* **extend** *algorithm, find the first (with respect to the lexicographical ordering) base image* $\gamma = (1, 2, \ldots, l-1, *, *, \ldots)$ *with* $\gamma > \beta$ *such that* γ *extends to some* $g \in G$. *If no such* γ *exists, replace* l *by* $l-1$, *else perform step*

(i) *Replace* β *by* γ. *If* $\beta^{(l)}$ *is first in its orbit, replace* S *by* $S \cup \{g\}$ *and replace the orbits by the new orbits under* $< S >$ *by merging any two orbits containing points* α, β *respectively for which* $\alpha^g = \beta$. *If* $\beta^{(l)}$ *is larger than every point that is first in an orbit, replace* l *by* $l-1$.

REMARKS. When constructing the automorphism group of a combinatorial structure, as in our example above, the overgroup H in which the search for G is conducted, is the full symmetric group S_Ω.

The *first in its orbit* criterion that is used in Step (2) is based on the following property relating g and its corresponding base image $(\gamma_1, \ldots, \gamma_k)$:

$$g \quad \text{is first in its left coset} \quad gH \quad \Longleftrightarrow \quad \forall i: \ \gamma_i \quad \text{is first in the orbit} \quad \gamma_i^{H^{(i)}}.$$

Usually it is too expensive to calculate $H^{(i)}$, because it involves a base change.

There is also a criterion for being first in the right coset:

$$g \quad \text{is first in its right coset} \quad Hg \quad \Longleftrightarrow \quad \forall i: \ \beta_i^g \quad \text{is first in} \quad \Delta_i^g.$$

Applying both criteria to base images simultaneously is an attempt to restrict consideration to one element from each double coset HgH.

Applications

In the application of backtrack search to permutation group problems, criteria specific to each particular type of problem are used to prune the search tree.

Two techniques often used in backtrack searches for subgroups of G satisfying a certain property, are the *choice of a suitable base*, and the *utilization of known subgroups*. Careful choice of the base enables us to apply our pruning criteria in an effective manner. The use of a known subgroup of the sought subgroup means that the *first in the coset* criteria are much more effective early in the search. Rather than explaining this in detail, we give a few examples of problems for which backtrack algorithms are commonly used. For details we refer the reader to [5, 8].

To find the *centralizer* $C_G(z) = \{g \in G: \ gz = zg\}$ of an element $z \in G$ one applies a base change to G so that the base consists of the first k points which appear in the initial cycles of z, (ideally the base is contained in the first cycle). Given the image of the first point in a cycle, the images of all other points in the cycle are determined: suppose that $\beta_{i+1} = \beta_i^z$ and that g centralizes z and let γ_j denote the image β_j^g; then $\gamma_{i+1} = \beta_{i+1}^g = \beta_i^{zg} = \beta_i^{gz} = \gamma_i^z$.

A slight modification yields an algorithm for determining the *centralizer* $C_G(H) = \{g \in G: \ gh = hg \text{ for } h \in H\}$ of a subgroup H of G. In this case one chooses a base compatible with the orbits of H. In particular, this provides a way of calculating the *centre* $Z(G) = \{g \in G: \ gx = xg \text{ for all } x \in G\}$.

Backtrack algorithms to determine the *normalizer* $N_G(H) = \{g \in g: \ gH = Hg\}$ of a subgroup H of G have also been developed.

Finding *set stabilizers* $G_{\{\alpha_1,\ldots,\alpha_r\}}$ is another application of backtrack methods. Although, as we mentioned, this is a hard problem (from a theoretical point of view), backtrack can be applied successfully. Here one can use the known subgroup technique, since the stabilizer of the set $S = \{\alpha_1, \ldots, \alpha_r\}$ will contain the pointwise stabilizer $G_{\alpha_1,\ldots,\alpha_r}$ as a subgroup. Using base change, if necessary, we choose a base B such that $\alpha_1, \ldots, \alpha_r$ appears as an initial segment: $B =$

$(\alpha_1, \ldots, \alpha_r, \beta_1, \ldots, \beta_s)$. The important thing is that now $G^{(r+1)}$ is a known subgroup of the stabilizer $G_{\{\alpha_1,\ldots,\alpha_r\}}$. As a consequence, only base images of the initial segment (rather than of the whole base) have to be considered, since each such image determines a coset of $G^{(r+1)}$ in $G_{\{\alpha_1,\ldots,\alpha_r\}}$, and the elements of such a coset either all belong to $G_{\{\alpha_1,\ldots,\alpha_r\}}$ or none do.

Finally, we mention a backtrack algorithm for finding the *intersection* $H_1 \cap H_2$ of two subgroups H_1 and H_2 of G. Here one chooses a common base for both subgroups, so that base images can be compared directly.

The BSGS backtrack search of a permutation group was introduced by Sims in 1970, when he described backtrack algorithms for computing centralizers and intersections of subgroups. Over the next decade, Butler, Cannon and Sims developed backtrack searches for constructing set stabilizers, normalizers, Sylow p-subgroups and for testing conjugacy of elements and subgroups (see [38], [39], [5], [6], [8]). HOLT [22] presents a backtrack algorithm for computing subgroup normalizers which employs many specialized tests to prune the backtrack search tree. The performance of his algorithm is superior in many cases to the Butler algorithm [6].

Very recent work of LEON [29] represents a major step in the evolution of backtrack algorithms for permutation groups. The efficiency of a backtrack search is heavily dependent upon the information available to prune the search tree. Using the idea of successive refinement of ordered partitions, first introduced by MCKAY [32, 33] as part of his highly successful graph isomorphism algorithm, Leon is able to devise new and powerful tests. An early implementation of a set stabilizer algorithm based on these ideas demonstrates performance that is dramatically superior to the "first generation" set stabilizer algorithm. As a result of this work we can expect a new generation of backtrack algorithms, exhibiting superior performance, to emerge in the near future.

6. HOMOMORPHISM METHODS

Homomorphisms

The use of structure-preserving mappings is a powerful tool in the design of algebraic algorithms. Each homomorphism ϕ of the permutation group G onto a group H is determined by a unique normal subgroup N of G (the kernel of ϕ). Unfortunately, if N is an arbitrary normal subgroup of G, then the quotient group G/N need not have a permutation representation of reasonable degree. Even if the quotient were known to possess a representation of small degree, it is not clear how one might construct this representation without first constructing the regular representation of G/N. Consequently, in practice, given an arbitrary normal subgroup N, we are restricted to constructing quotients of G by N in situations where $[G : N]$ is less than a million. This situation is partially relieved by the fact that there are two important kinds of permutation group homomorphisms that permit very efficient computation.

Firstly, as noted in Section 2, the restriction of the action of G to a G-invariant subset Δ of Ω defines a homomorphism of G into a subgroup G^{Δ} of the symmetric

group S_Δ; we will call such a homomorphism a *constituent homomorphism.* Secondly, if Γ is a G-invariant partition of Ω (i.e. a system of imprimitivity for G in its action on Ω), the action induced by G on the sets of Γ, defines a homomorphism of G into a subgroup G^Γ of the symmetric group S_Γ. Such a homomorphism will be called a *blocks homomorphism.* The combination of these two homomorphisms enables us to perform a crucial reduction. Taking Δ to be an orbit of G, we obtain a transitive group $H = G^\Delta$, acting on the set Δ. Now taking Γ to be a maximal proper H-invariant partition of Δ, we obtain a primitive group $K = H^\Gamma$ acting on Γ.

Suppose that a BSGS is known for G. Let $\phi : G \to H$ be a homomorphism. In order to compute effectively with ϕ we need to have efficient algorithms to solve the following problems:

(a) Construct a BSGS for imϕ and kerϕ directly from that for G.

(b) Determine images $\phi(g)$ for $g \in G$ and preimages $\phi^{-1}(h)$ for $h \in H$.

(c) Construct a BSGS for $\phi(I)$, with $I \leq G$ and for $\phi^{-1}(K)$, with $K \leq H$, from the given ones for I and K.

Efficient solutions for the constituent and blocks homomorphisms were developed in the 1970's by Butler and Cannon (see Butler [7]). Let us first consider the case where ϕ is a constituent homomorphism.

ALGORITHM CONSTITUENT HOMOMORPHISM

Input Ω; permutation group G acting on Ω equipped with BSGS; a G-invariant subset Δ of Ω.

Output Bases $B_{\ker}$, B_{im} and strong generating sets $S_{\ker}$, S_{im} for kerϕ and imϕ of the constituent homomorphism ϕ.

(1) *Choose points $\alpha_1, \ldots, \alpha_r$ of Δ such that $G_{\alpha_1,\ldots,\alpha_r}$ is the pointwise stabilizer of Δ in G; choose points $\beta_1, \ldots, \beta_s$ of Ω such that $B = (\alpha_1, \ldots, \alpha_r, \beta_1, \ldots, \beta_s)$ is a base for G.*

(2) *Apply the base change algorithm to obtain a strong generating set S for G relative to B.*

(3) *Put $B_{\mathrm{im}} = (\alpha_1, \ldots, \alpha_r)$ and $S_{\mathrm{im}} = \phi(S \setminus S^{(r+1)})$.*

(4) *Put $B_{\ker} = (\beta_1, \ldots, \beta_s)$ and $S_{\ker} = S^{(r+1)}$.*

REMARKS. Thus, for the constituent homomorphism, problem (a) is solved by changing the base for G so that the points of Δ appear at the beginning. Computing the image of an element is then trivial. Now consider an element $h \in \mathrm{im}\phi$. Viewing Δ as a subset of Ω and using the BSGS set for G, construct an element $g \in G$ such that $(\alpha_1, \ldots, \alpha_r)^g = (\gamma_1, \ldots, \gamma_r) = (\alpha_1, \ldots, \alpha_r)^h$. Since

$\phi^{-1}(h) = \ker\phi \cdot g$, we have solved problem (b). To obtain a BSGS for the image of a subgroup I of G, one proceeds exactly as in the above algorithm, with G replaced by I. Finally, to obtain a BSGS for the preimage of a subgroup K of H, one changes base to obtain strong generators T relative to the base $B = (\alpha_1, \ldots, \alpha_r)$, and forms the preimage for each element of T as above. Then $\phi^{-1}(T) \cup S^{(r+1)}$ is a strong generating set for $\phi^{-1}(K)$ with respect to B.

We now consider the case where ϕ is a blocks homomorphism. Assume that G acts transitively on the set Ω and suppose that $\Gamma = \{\Psi_1, \ldots, \Psi_r\}$ is a system of imprimitivity for G. It is convenient at this point to consider the action of G on the set $\Omega' = \Gamma \cup \Omega$. Applying the previous lemma with $\Omega = \Omega'$, $\Delta = \Gamma$ and $\alpha_i = \Psi_i$ for $i = 1, \ldots, r$, immediately gives us a BSGS for $\mathrm{im}\phi$ and for $\ker\phi$. The only complication is that we have to compute the chain of stabilizers for the blocks:

$$G \geq G_{\Psi_1} \geq G_{\Psi_1,\Psi_2} \geq \cdots \geq G_{\Psi_1,\ldots,\Psi_r}$$

in place of the chain of sequence stabilizers

$$G \geq G_{\alpha_1} \geq G_{\alpha_1,\alpha_2} \geq \cdots \geq G_{\alpha_1,\ldots,\alpha_r}.$$

Although the construction of the stabilizer of an arbitrary set is known to be hard (as we saw), the stabilizer of a block may be computed efficiently using the following simple algorithm.

ALGORITHM BLOCK STABILIZER

Input Ω; permutation group G acting on Ω, equipped with BSGS; a block Ψ.

Output Generators for the block stabilizer G_Ψ.

(1) *Choose $\alpha \in \Psi$; use the point stabilizer Algorithm to obtain generators X for G_α. Put $Y = \{\mathrm{id}\}$.*

(2) *Calculate the orbit α^G. Perform Step (a) until $\alpha^{<Y>} = \Psi \cap \alpha^G$,*

  (a) *Choose $\gamma \in (\Psi \cap \alpha^G) \setminus \alpha^{<Y>}$ and find $g \in G$ such that $\alpha^g = \gamma$. Replace Y by $Y \cup \{g\}$.*

(3) *Return $X \cup Y$.*

REMARKS. The correctness of this algorithm is based the following observation [8]: if X is a generating set for G_α with $\alpha \in \Psi$, and Y is a subset of G such that $\alpha^{<Y>} = \Psi \cap \alpha^G$, then $G_\Psi = < X \cup Y >$ (because $< X \cup Y >$ is transitive on $\Psi \cap \alpha^G$ and $< X >_\alpha = G_\alpha$).

ALGORITHM BLOCKS HOMOMORPHISM

Input Ω; permutation group G acting transitively on Ω, equipped with BSGS; a system $\Gamma = \{\Psi_1, \ldots, \Psi_r\}$ of imprimitivity for G.

Output Bases $B_{\ker}$, Ψ_{im} and strong generating sets $S_{\ker}$, S_{im} for $\ker\phi$ and $\mathrm{im}\phi$ of the constituent homomorphism ϕ.

(1) *Choose points* $\beta_1, \ldots, \beta_s$ *of* Ω *such that* $B = (\Psi_1, \ldots, \Psi_r, \beta_1, \ldots, \beta_s)$ *is a base for* G *in its action on* $\Gamma \cup \Omega$.

(2) *Apply base change and the Block Stabilizer Algorithm to obtain a strong generating set* S *for* G *relative to* B.

(3) *Put* $B_{\mathrm{im}} = (\Psi_1, \ldots, \Psi_r)$ *and* $S_{\mathrm{im}} = \phi(S \setminus S^{(r+1)})$.

(4) *Put* $B_{\ker} = (\beta_1, \ldots, \beta_s)$ *and* $S_{\ker} = S^{(r+1)}$.

REMARKS. This provides a solution to problem (a) for the blocks homomorphism. Images and preimages are calculated in a manner similar to constituent homomorphism, with a slight complication caused by the fact that we have to translate between the action of G on $\Gamma \cup \Omega$ and its action on Ω.

Sylow subgroups

We now apply these ideas to design an algorithm for constructing the Sylow p-subgroup P of a permutation group G.

There have been two distinct approaches to the design of practical algorithms for this problem. One method involves constructing P by means of successive cyclic extensions. An initial subgroup P_1 of P is sought by examining a random selection of elements of G for one having order divisible by p. Suppose the subgroup P_i of P has been constructed. In order to construct P_{i+1}, we require an element $x \in G \setminus P_i$ such that x normalizes P_i and $x^p \in P_i$. Rather than searching the whole of G for such an element we may restrict our search to some suitably chosen subgroup T of G. Thus, Cannon [13] and Butler and Cannon [10] choose T to be $C_G(z)$ for an element z in the centre $Z(P)$ such that $|\mathrm{sylow}_p(C_G(z))| > |P_i|$, while Holt [22] chooses T to be $N_G(P_i)$. In either case a backtrack search is employed to construct T (see Section 5). The Butler-Cannon algorithm then uses a second backtrack procedure to search T for a suitable extending element x. These backtrack-based methods become rather inefficient once the exponent of p in G exceeds 15 or 16.

The second approach utilizes the following lemma.

LEMMA. *Let* P *be a* p*-group acting on the set* Ω*, with orbits* $\Delta_1, \ldots, \Delta_r$*. Let* K *be the kernel of the natural action of* $C_{S_\Omega}(P)$ *on the set* $\{\Delta_1, \ldots, \Delta_r\}$*. Then* K *is a* p*-group.*

This approach was suggested by P. Neumann in 1983 and was subsequently developed into a practical algorithm by Butler and Cannon [10].

ALGORITHM SYLOW

Input Ω; a permutation group G acting on Ω with known BSGS; a prime number p.

Output Sylow p-subgroup P of G.

(0) *Put* $P_0 = \mathrm{id}$. *If* p *does not divide the order of* G, *put* $P = P_0$ *and terminate.*

(1) *Put* $i = 1$, *and find* $P_1 = < x >$ *by generating random elements* y *of* G *until* y *is found with order* o *divisible by* p, *whereupon* $x = y^{m/p^h}$, *with* p^h *being the largest power of* p *dividing* m.

Repeat the following steps until termination.

(2) *Locate an element* $z \neq 1$ *in* $Z(P_i)$ *such that* $|\mathrm{sylow}_p(C_G(z))| > |P_i|$; *if no such element can be found,* $P = P_i$ *and terminate. Find* $C = C_G(z)$. *Further, let* Γ *be the fixed point set of* C *and let* $\Delta = \Omega \setminus \Gamma$.

(3) *Construct the constituent homomorphisms* $\phi : C \to C^{\Delta}$ *and* $\psi : C \to C^{\Gamma}$.

(4) *Let* Σ *denote the partition of* Δ *defined by the cycles of* $\phi(z)$. *Construct the blocks homomorphism* $\sigma : C^{\Delta} \to (C^{\Delta})^{\Sigma}$. *Recursively compute the Sylow* p*-subgroup* $\bar{Q}$ *of* $\sigma(\phi(C))$. *Let* Q *be the preimage of* $\bar{Q}$ *in* C, *so* $Q = \phi^{-1}(\sigma^{-1}(\bar{Q}))$.

(5) *Recursively compute the Sylow* p*-subgroup* $\bar{R}$ *of* $\psi(Q)$. *Put* $P_{i+1} = \psi^{-1}(\bar{R})$.

(6) *Replace* i *by* $i + 1$.

REMARKS. Note that before applying the lemma, we first restrict the centralizer C so as to act solely on the p-cycles. Experience has shown that this additional reduction of the degree leads to better performance. If it is possible to quickly locate an element z of order p in the centre of $\mathrm{sylow}_p(G)$, then rather than constructing our way up a chain of subgroups to P, we may obtain P directly by taking C in the above procedure to be $C_G(z)$.

Experimentally, it has been found that this algorithm generally runs a great deal faster than the backtrack method. Another method based on the use of homomorphisms has been suggested by ATKINSON and NEUMANN [2].

KANTOR [24] and KANTOR and TAYLOR [26] show that the Sylow p-subgroup of a finite permutation group may be found in polynomial time. However, their proof does not appear to provide the basis for a practical algorithm.

7. COMPOSITION SERIES

Composition series of a permutation group

A *composition series* for G is a series

$$\{\mathrm{id}\} = H_0 \triangleleft H_1 \triangleleft H_2 \triangleleft \cdots \triangleleft H_r = G$$

of *subnormal* subgroups H_i such that each quotient H_i/H_{i-1} is simple ($1 \leq i \leq r$). (These subgroups are called subnormal because while H_i is normal in H_{i+1}, it is not necessarily normal in G.)

Suppose G is an arbitrary permutation group. Using the techniques described in section 6 we may construct a subnormal series as above such that each quotient H_i/H_{i-1} is primitive ($1 \leq i \leq r$). Thus, the problem of constructing a composition series for G has been reduced to the problem of constructing a composition series for a primitive group. However, the possibilities for the structure of a primitive group are severely constrained by the O'Nan-Scott Theorem, as we will see below. Analyzing the structure of G using this theorem will require a base and strong generating set; but before constructing these it is highly desirable to recognize the case in which G contains the alternating group A_n.

The detection of A_n as a subgroup

Assume that G acts faithfully on the set Ω of cardinality n. We wish to devise a fast algorithm for deciding whether or not the alternating group A_n is a subgroup of G. Of course, if a BSGS is already known for G, the question may be settled instantly by examining the order of G. However, if the degree of G is large, it is very important to detect the presence of A_n before attempting to construct a BSGS for G. We outline a fast Las Vegas algorithm for this problem. Upon termination, our algorithm will have either proved that $A_n \leq G$, or have established that it is unlikely that $A_n \leq G$.

The starting point for our algorithm is the following theorem due to Jordan.

THEOREM. *Let G be a primitive permutation group of degree n containing a p-cycle, where p is a prime such that $p \leq n-3$. Then $A_n \leq G$.*

Thus, if a primitive permutation group contains a cycle x of prime length fixing at least three points, then $A_n \leq G$; numerous workers have extended this result to the case where x is a product of q cycles of length p, with p prime and $1 \leq q < p$ (see WIELANDT [41, II.13]). The most general theorem of this type was proved by PRAEGER [36]. Before stating the theorem it is convenient to introduce the following notation. A *Jordan element* for a primitive permutation group G of degree n is an element of prime order p consisting of q cycles of length p (with $1 \leq q < p$) and $k = n - qp$ fixed points. A Jordan element which cannot belong to any smaller subgroup of S_n than A_n will be called a *Jordan witness* (for A_n).

THEOREM (PRAEGER). *Suppose G is a primitive permutation group of degree n for which $A_n \not\leq G$ and which contains a Jordan element. Suppose further that $n \neq m(m-1)/2$ (for any m) and $n \neq q$.*

Then $k \leq 5q/2 - 2$.

For small values of q better bounds on k are known (see [41, II.13]). Following Praeger's result, LIEBECK and SAXL [30] produced a complete classification of all groups containing a Jordan element for which $A_n \not\leq G$. Taken together, Jordan's

theorem and its generalizations by Manning, Praeger, Liebeck and Saxl allow one to determine whether a Jordan element is in fact a Jordan witness.

We now outline a variant of the algorithm [12], in which the construction of a BSGS is started if A_n is not detected by the probabilistic part of the algorithm. In Step (3), the fact that all 2-transitive groups are known ([11, 12]) is used; thus, in both (2) and (3), the classification of finite simple groups is used. (This assumption may be avoided at some cost in efficiency).

ALGORITHM DETECT ALTERNATING

Input Ω; a permutation group G acting on Ω; a positive integer m.

Output True if $A_n \leq G$, false otherwise.

(1) *Apply the primitivity test of section 2. If G is found to be imprimitive, note that $A_n \not\leq G$ and terminate.*

(2) *Generate m 'random' elements of G and examine their powers for the presence of Jordan elements. Determine whether any of these Jordan elements is actually a Jordan witness. If such a witness is found, note that $A_n \leq G$ and terminate.*

(3) *Construct a BSGS for G. If, during its construction, G is found to be t-transitive ($t \geq 2$) and if the degree of G does not correspond to that of a t-transitive group other than A_n or S_n, abort the construction and report that $A_n \leq G$. If the construction proceeds to completion, report whether or not $A_n \leq G$ (by calculating the order) and terminate.*

REMARKS. If a suitable value has been chosen for m, the probability that $A_n \leq G$ will be small once one gets to the stage where a BSGS has to be constructed. The trick with this algorithm is to choose m sufficiently small so that the cost of the algorithm is not excessive when $A_n \not\leq G$, but sufficiently large so that the probability of initiating the BSGS construction on a group containing A_n is small.

What is the probability of finding a Jordan witness among a sample of m elements chosen from G? The following result of P. Neumann [34] provides a partial answer; see also [19].

LEMMA. *Let $j(n)$ denote the number of elements in S_n which contain exactly one cycle of length p, with $p \leq n-3$ prime, and are such that the lengths of all the other cycles are not divisible by p. Then:*

$$\frac{j(n)}{n!} - \frac{1}{\log_2(n)} \geq \epsilon, \text{for a non-negative function } \epsilon \in O((\log_2 n)^{-2}).$$

Since random elements are also used in the primitivity test, in practice, the test for primitivity and the search for a Jordan witness are performed in parallel.

The major technical difficulty with the above algorithm concerns the need to generate randomly chosen elements of G. The only way of producing elements in G is to evaluate words over the generating set X. However, it is not difficult to show that in order to generate unbiased elements it is necessary to choose extremely long words. If this is done, the cost of the algorithm is prohibitive. In practice, therefore, we are forced to work with relatively short words in the generators. Consequently, the elements of G that we examine may not represent a truly random sample.

Example One hundred groups generated by pairs of random permutations on $100,000$ letters were constructed. The algorithm correctly determined that the alternating group of degree $100,000$ was a subgroup of each of these groups. The mean recognition time for the 100 trials was 24 seconds with a standard deviation of 20. Repeating the trial with permutations of degree $1,000,000$, gave a mean recognition time of 296 seconds with standard deviation 240.

The O'Nan-Scott Theorem
Let G be a primitive group on the set Ω. Let N be a minimal normal subgroup of G. Then N is transitive and its centralizer $C_G(N)$ is also a normal subgroup of G. There are two cases depending upon $C_G(N)$:

(i) If $C_G(N) \neq 1$, then $C_G(N)$ is transitive, so that both N and $C_G(N)$ are regular. Note that if N_1 and N_2 are distinct minimal normal subgroups of G, then $N_2 \leq C_G(N_1)$ and $N_1 \leq C_G(N_2)$. Thus, if N is abelian, it is the unique minimal normal subgroup of G, while if it is non-abelian, G has exactly two minimal normal subgroups (N and $C_G(N)$). Further, since N and $C_G(N)$ are left regular and right regular representations of the same group, they are isomorphic.

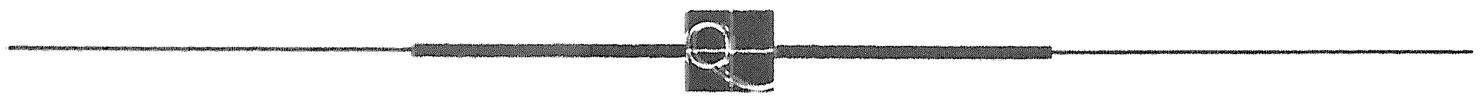
(ii) If $C_G(N) = 1$, then N is non-abelian and it is the unique minimal normal subgroup of G, and G is isomorphic to a subgroup of the automorphism group of N.

Recalling the fact that a minimal normal subgroup of a finite group is a direct product of isomorphic simple groups we see that, in every case, the *socle* of G (the product of its minimal normal subgroups) is a direct product of isomorphic simple groups. A somewhat more detailed analysis of the possible structure of a primitive group is provided by the O'Nan-Scott Theorem [11, 24, 35].

THEOREM (O'NAN-SCOTT). *Let G be a primitive group acting faithfully on a set Ω of cardinality n. Then one of the following cases holds:*

(I) *G has an elementary abelian regular normal subgroup of order $n = p^d$, for some prime p and some $d \geq 1$.*

(II) *The socle of G is a direct product $N = T_1 \times \cdots \times T_k$ of isomorphic non-abelian simple groups T_i. In this case one of the following must hold:*



(i) *Ω can be identified with a set $\bar{\Omega} \times \cdots \times \bar{\Omega}$ (k copies), so that $n = \bar{n}^k = |\bar{\Omega}|^k$, and the action of G is the wreathed product action; moreover there is a faithful primitive permutation representation on $\bar{\Omega}$ of a group containing T_1 as a normal subgroup.*

(ii) *$n = |T_1|^{(a-1)b}$, for integers a and b with $ab = k > 1$, and $N_\alpha = D_1 \times \cdots \times D_b$ for $\alpha \in \Omega$, where D_i is a diagonal subgroup of $T_{(i-1)a+1} \times \cdots \times T_{ia}$; furthermore G acts transitively on $\{T_1, \ldots, T_k\}$ with block system $\{\{T_{(i-1)a+1}, \ldots, T_{ia}\} : i = 1, \ldots, b\}$.*

(iii) *$n = |T_1|^k$, with $k \geq 2$, and G acts transitively on $\{T_1, \ldots, T_k\}$.*

(iv) *$n = |T_1|^{\frac{k}{2}}$, with $k \geq 4$, and G has two orbits on $\{T_1, \ldots, T_k\}$ of length $k/2$.*

Abelian socle

From the analysis above we see that an algorithm is required for constructing an elementary abelian regular normal subgroup of G (if it exists). We suppose that $n = p^d$, for some prime p. Let us denote an elementary abelian regular normal subgroup of G (if it exists) by N. Now N is equal to the largest elementary abelian normal p-subgroup $O_p(G)$ of G. One approach is to calculate $O_p(G)$ by computing the Sylow p-subgroup P using the method of Section 6, and then intersecting P with its conjugates (using the intersection algorithm of Section 5). However, for $d > 20$ it becomes expensive to compute P and so we seek an alternative approach to the computation of N.

The approach we describe here is due to P. Neumann and involves constructing a small set of elements, one of which is guaranteed to lie in N. Then N may be obtained by applying the normal closure algorithm of Section 4 to the subgroup generated by each of these elements in turn. Since N is very small, an application of the normal closure algorithm in this context is cheap.

The starting point is the following easy lemma, see Neumann [34].

LEMMA. *Let G be a permutation group containing a regular normal subgroup N. Let H be a subgroup of G with fixed point set Γ, where $|\Gamma| \geq 2$, and let C be the centralizer of H in G. Then:*
(i) *C acts transitively on Γ, and*
(ii) *$C \cap N$ acts regularly on Γ.*

It is obviously desirable to choose the subgroup H in such a way that its centralizer is as small as possible. Consequently, we take H to be a two point stabilizer in G. In fact, by taking H to be the stabilizer of the first two base points β_1, β_2, we obtain a suitable H without the need for any additional computation. The general structure of the elementary abelian regular normal subgroup algorithm is as follows.

ALGORITHM ELEMENTARY ABELIAN REGULAR NORMAL SUBGROUP

Input Ω; a primitive permutation group G on Ω equipped with a BSGS.

Output An elementary abelian regular normal subgroup N for G if it exists.

(1) *Let p be prime such that $n = p^d$; if such p does not exist, then G does not have an elementary abelian regular normal subgroup: terminate.*

(2) *If G is regular, then set $N = G$ and terminate.*

(3) *If G is a Frobenius group, then calculate the Frobenius kernel and terminate (this is easy since the order of G is small).*

(4) *Let β_1, β_2 be the first two base points of G; we may assume $H = G_{\beta_1,\beta_2} \neq \{\mathrm{id}\}$. Calculate the fixed point set Γ of H. If $|\Gamma|$ is not a power of p, then terminate, since G does not have an elementary abelian regular normal subgroup.*

(5) *Calculate $C = C_G(H)$; if C does not act transitively on Γ, then terminate since G does not have an elementary abelian regular normal subgroup.*

(6) *Calculate $P = O_p(C^\Gamma)$; if P does not act transitively on Γ, then terminate since G does not have an elementary abelian regular normal subgroup.*

(7) *Calculate the centre Z of $O_p(C)$.*

(8) *For each $z \in Z$ for which $< z >$ acts regularly on Ω, compute $N = ncl_G(< z >)$. If some N is regular, return N and terminate; if no element of Z generates a regular normal subgroup, then G does not possess an elementary abelian regular normal subgroup.*

If G does contain an elementary abelian regular normal subgroup N, it is a straightforward matter to construct a composition series $\{\mathrm{id}\} = N_0 \lhd N_1 \lhd N_2 \lhd \cdots \lhd N_r = N$ for N. We recursively compute a composition series $\{\mathrm{id}\} = G_0 \lhd G_1 \lhd G_2 \lhd \cdots \lhd G_r$ for the stabilizer G_α of a point in G. The series

$$\{\mathrm{id}\} = N_0 \lhd N_1 \lhd N_2 \lhd \cdots \lhd N_r = N \lhd NG_1 \lhd NG_2 \lhd \cdots \lhd NG_r = G$$

is then a composition series for G.

REMARKS. Since (β_1, β_2) is a base for C^Γ, we have $|C^\Gamma| \leq |\Gamma|(|\Gamma| - 1)$. Consequently, C^Γ is a very small group and $O_p(C^\Gamma)$ may be computed using the Sylow p-subgroup algorithm.

Note that $O_p(C)$ may be obtained at little cost. The kernel of the homomorphism $\phi : C \to C^\Gamma$ is $C \cap G_{\beta_1,\beta_2}$, which is the same as $Z(G_{\beta_1,\beta_2})$. Then $O_p(C)$ is the p-primary part of the nilpotent group $< Z(G_{\beta_1,\beta_2}), R >$, where R is the subgroup of C generated by the preimages of a set of generators for $O_p(C^\Gamma)$.

For further details the reader should consult [34].

EXAMPLES.

(1) Consider $G = S_{32} \wr S_2$ which has degree 1024 and order

$$1384756746908520303863323978879191673089370263814838157312 \cdot 10^{14}.$$

Taking β_1, β_2 to be the first two points of the base computed for G, we obtain a set Γ of cardinality 2 and a subgroup C having order 2. Exclusive of the time taken to construct a BSGS for G, the algorithm, as implemented in Cayley, establishes that G has no elementary abelian regular normal subgroup in 100 seconds. This time is completely dominated by the 96 seconds required to compute $C_G(G_{\beta_1,\beta_2})$.
(2) Consider G =AGL(12, 2) which has degree 4096 and order

$$263854583512504817331360558342180020850524160000.$$

Again the set Γ has cardinality 2 while the subgroup C has order 2. Starting with a known BSGS for G, the algorithm constructed the elementary abelian regular normal subgroup in 490 seconds. Of this, 350 seconds were spent constructing the centralizer $C_G(G_{\beta_1,\beta_2})$, and 105 seconds were spent forming the normal closure which produced the required normal subgroup.

Non-abelian socle
We now assume that case (II) of the O'Nan-Scott theorem applies. The socle of G may be found in this case with the aid of the following theorem.

THEOREM. *Let G be a finite group such that $O_p(G)$ is trivial for each prime p that divides $|G|$. If N is the normal closure of $Z(\text{sylow}_2(G))$, then the socle of G is equal to the solvable residual $N^{(\infty)}$ of N.*

For a proof see [34].

This theorem reduces the calculation of the socle to the application of standard algorithms. However, in the range of permutation group degrees amenable to practical computation, the O'Nan-Scott theorem allows us to do a great deal better.

LEMMA. (Neumann) *Let G be a primitive permutation group of degree n. If $n < 60^4 = 12,960,000$, and n cannot be expressed in the form m^d, with integers $m \geq 5$ and $d \geq 5$, then the socle of G is $G^{(\infty)}$.*

For $n \leq 10^6$ this lemma covers all but 21 possible degrees. Thus, our socle algorithm uses this lemma if applicable, and otherwise resorts to the previous theorem. For groups of degree $n = m^d < 10^6$ (with $m, d \geq 5$), Kantor [25] describes an alternative approach to the calculation of the socle.
The final step involves *splitting* the socle N of G into its simple direct factors. We know that either

$$N = M_1 \qquad \text{or} \qquad N = M_1 \times M_2$$

with $M_1 = T_1 \times \cdots \times T_d$ and $M_2 = T_{d+1} \times \cdots \times T_{2d}$, where M_1 and M_2 are normal in G and where $T_1 \cong \ldots \cong T_d \cong T_{d+1} \cong \ldots \cong T_{2d} \cong T$, a non-abelian simple

group. Let Γ be a maximal proper block system for N in its action on Ω and let ψ be the homomorphism $\psi : N \to N^\Gamma$. Then N^Γ is primitive and so is isomorphic to N/Y, where Y is a maximal normal subgroup of N. Now, renumbering if necessary, $Y = T_2 \times \cdots \times T_d$ (if $N = M_1$), or $Y = T_2 \times \cdots \times T_d \times T_{d+2} \times \cdots \times T_{2d}$ (if $N = M_1 \times M_2$). So for $X = C_N(\ker\psi)$ we have $X = T_1$ (if $N = M_1$), or $X = T_1 \times T_{d+1}$ (if $N = M_1 \times M_2$). This suggests the following algorithm.

ALGORITHM SPLIT SOCLE

Input Ω; a primitive permutation group G on Ω equipped with a BSGS; the socle N of G.

Output The simple direct factors of N.

(1) *Construct a maximal proper block system Γ for N. Let Y be the kernel of the action of N on Γ, and let $X = C_N(Y)$.*

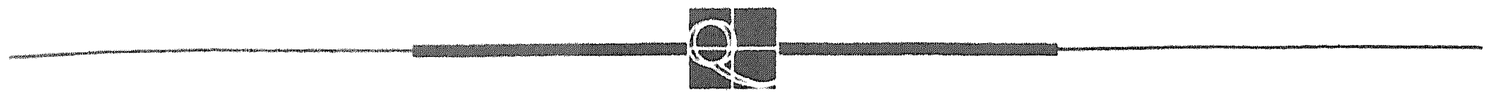
(2) *If $|X| \neq |\Gamma|^2$, then $N = M_1$ and the factors $T_2, \cdots, T_d$ are obtained by conjugating $X = T_1$ repeatedly by the generators of G.*

(3) *Construct $Z = Z(C_X(x))$, for $x \neq 1$, $x \in X$. If for some $z \in Z$, $z \neq 1$, the normal closure $X_1 = ncl(< z >)$ is a proper subgroup of X, then $N = M_1 \times M_2$, so set $T_1 = X_1$, and $T_d = C_X(X_1)$; the remaining direct factors are obtained as the G-conjugates of T_1 and T_d. Otherwise, $N = M_1$, so the remaining direct factors $T_2, \ldots, T_d$ are constructed as the G-conjugates of T_1.*

REMARKS. Let K be the kernel of the action of G on the simple factors of N. A composition series for G now consists of the obvious composition series for N, followed by subgroups of K which, modulo N, form a composition series for the abelian group K/N, followed by subgroups of G which, modulo K, form a composition series for the (small) group G/K.

EXAMPLE. The wreath product (with product action) of the projective group PGL(2, 9) and the symmetric group S_5, is a primitive group of degree 100,000 and order 2321901584000000. Using a variant of the composition series algorithm that identifies the composition factors (Kantor [25]), we obtain the following picture of the abstract group structure:

```
COMPOSITION FACTORS OF GROUP K
------------------------------
                G
                |
            Cyclic(2)
                *
                |
            Cyclic(2)
                *
                |
          Alternating(5)
```



```
*
| Cyclic(2)
*
| Cyclic(2)
*
| Cyclic(2)
*
| Alternating(6)
*
| Alternating(6)
*
| Alternating(6)
*
| Alternating(6)
*
| Alternating(6)
1
```

Including the time taken to construct a BSGS, the calculation took approximately 5800 seconds. Most of the time was spent computing the derived subgroup.

References

1. M. D. Atkinson (1974). An algorithm for finding the blocks of a permutation group. *Math. Comp.* 29, 911–913.
2. M. D. Atkinson, P. M. Neumann (1990). Computing Sylow subgroups of permutation groups. *Congressus Numerantium* 79, 55–60.
3. L. Babai, G. Cooperman, L. Finkelstein, E. Luks, Á. Seress (1991). Fast Monte Carlo algorithms for permutation groups. *Proc. 23rd Ann. ACM Symp. on Theory of Computing (STOC)*, ACM-Press, 90–100.
4. C. A. Brown, L. Finkelstein, P. W. Purdom (1989). A new base change algorithm for permutation groups. *SIAM J. Computing* 18, 1037–1047.
5. G. Butler (1983). Computing in permutation and matrix groups II: backtrack algorithm. *Math. Comp.* 39, 671–680.
6. G. Butler (1983). Computing normalizers in permutation groups. *J. Algorithms* 4, 163–175.
7. G. Butler (1985). Effective computation with group homomorphisms. *J. Symbolic Computation* 1, 143–157.
8. G. Butler (1991). Fundamental Algorithms for Permutation Groups. *Lecture Notes in Computer Science* 559, Berlin: Springer.
9. G. Butler, J. J. Cannon (1989). Computing in permutation and matrix groups III: Sylow subgroups. *J. Symbolic Computation* 8, 241–252.

10. G. Butler, J. J. Cannon (1991). Computing Sylow subgroups using homomorphic images of centralizers, *J. Symbolic Computation* 12, 443–457.
11. P. J. Cameron (1981). Finite permutation groups and finite simple groups, *Bull. London Math. Soc.* 13, 1–22.
12. P. J. Cameron, J. J. Cannon, *Fast recognition of alternating and symmetric groups*. In preparation.
13. J. J. Cannon (1971). Computing local structure of large finite groups. G. Birkhoff, M. Hall, Jr. (eds.), *Computers in Algebra and Number Theory*, SIAM-AMS Proceedings 4, 161–176.
14. J. J. Cannon (1984). An introduction to the group theory language Cayley. M.D. Atkinson (ed), *Computational Group Theory*, Proceedings of the LMS Symposium, Durham, July 30–August 9, 1982, London: Academic Press, 143–182.
15. J. J. Cannon (1984). A computational toolkit for finite permutation groups. *Proceedings of the Rutgers Group Theory Year 1983–1984*, New York: Cambridge University Press, 1–18.
16. G. Cooperman, L. Finkelstein (1991). A strong generating test and short presentations for permutation groups. *J. Symbolic Computation* 12, 475–497.
17. G. Cooperman, L. Finkelstein, E. Luks (1989). Reduction of group constructions to point stabilizers. G. Gonnet (ed.) *Proceedings of the 1989 International Symposium on Symbolic and Algebraic Computation (ISSAC '89)*, New York: Assoc. Comput. Mach., 351–356.
18. G. Cooperman, L. Finkelstein, N. Sarawagi (1990). A random base change algorithm for permutation groups. S. Watanaba, M. Nagata (eds), *Proceedings of the 1990 International Symposium on Symbolic and Algebraic Computation (ISSAC '90)*, New York: Assoc. Comput. Mach., 161–168.
19. J. D. Dixon (1969). The probability of generating the symmetric group. *Math. Z.* 10, 199–205.
20. M. Furst, J. E. Hopcroft, E. Luks (1980). Polynomial time algorithms for permutation groups. *Proc. 21st IEEE Foundations of Computer Science*, 36–41.
21. G. Havas (1991). Coset enumeration strategies. S. M. Watt (ed.), *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC '91)*, New York: Assoc. Comput. Mach., 191–199.
22. D. F. Holt (1991). The computation of normalizers in permutation groups. *J. Symbolic Computation* 12, 499–516.
23. M. R. Jerrum (1986). A compact representation for permutation groups. *J. Algorithms* 7, 60–78.
24. W. M. Kantor (1985) Sylow's theorem in polynomial time. *J. Comput. System Sci.* 30, 359–394.
25. W. M. Kantor (1991). Finding composition factors of permutation groups of degree $n \leq 10^6$, *J. Symbolic Computation* 12, 517–526.
26. W. M. Kantor, D. E. Taylor (1988). Polynomial-time versions of Sylow's theorem. *J. Algorithms* 9, 1–17.
27. D. E. Knuth (1991). Notes on efficient representation of perm groups. *Com-*

binatorica 11, 33–43.
28. J. S. LEON (1980). On an algorithm for finding a base and strong generating set for a group given by generating permutations. *Math. Comp.* 35. 941–974.
29. J. S. LEON (1991). Permutation group algorithms based on partitions I: Theory and algorithms. *J. Symbolic Computation* 12. 533–583.
30. M. W. LIEBECK, J. SAXL (1985). Primitive permutation groups containing an element of large prime order. *J. London Math. Soc.* 31. 237–249.
31. E. LUKS (1982). Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. Sys. Sci.* 25. 42–65.
32. B. D. MCKAY (1978). Computing automorphisms and canonical labellings of graphs. D. A. HOLTON, J. SEBERRY (eds), *Combinatorial Mathematics*, Lecture Notes in Mathematics 686. Berlin: Springer.
33. B. D. MCKAY (1981). Practical graph isomorphism. *Congressus Numerantium* 30, 45–87.
34. P. M. NEUMANN (1987). Berechnungen in Gruppen. *Lecture notes*, Mathematics Department ETH Zürich.
35. P. M. NEUMANN (1986). Some algorithms for computing with finite permutation groups. E. F. ROBERTSON, C. M. CAMPBELL (eds.), *Groups - St. Andrews 1985*, London Math. Soc. Lecture Notes Series 121, Cambridge: Cambridge University Press, 59–92.
36. C. E. PRAEGER (1979). On elements of prime order in primitive permutation groups, *J. Algebra* 60. 126–132.
37. C. C. SIMS (1970). Computational methods in the study of permutation groups. J. Leech (ed.), *Computational Problems in abstract algebra*, Oxford: Pergamon, 169–183.
38. C. C. SIMS (1971). Determining the conjugacy classes of a permutation group. G. BIRKHOFF, M. HALL, Jr. (eds.), *Computers in Algebra and Number Theory*, SIAM-AMS Proceedings 4, Providence: Amer. Math. Soc., 191–195.
39. C. C. SIMS (1971). Computation with permutation groups. S. R. PETRICK (ed.), *Proc. Second Symp. on Symbolic and Algebraic Computation*, New York: Assoc. Comp. Mach., 23–28.
40. J. A. TODD, H. S. M. COXETER (1936). A practical method for enumerating cosets of a finite abstract group. *Proc. Edinburgh Math. Soc.* 5, 26–34.
41. H. WIELANDT (1964). *Finite permutation groups*, New York: Academic Press.