The Schreier-Sims algorithm

An essay
submitted to the Department of mathematics,
Australian National University,
in partial fulfillment of the requirements
for the degree of
Bachelor of Science with Honours

By
Scott H. Murray
November 1993

Acknowledgements

I would like to thank my supervisor Dr E.A. O'Brien, who has been a constant source of help and encouragement.

I am also grateful to Dr John Cannon of Sydney University and Professor Cheryl E. Praeger of the University of Western Australia for enlightening discussions on aspects of this project.

Abstract

A base and strong generating set provides an effective computer representation for a permutation group. This representation helps us to calculate the group order, list the group elements, generate random elements, test for group membership and store group elements efficiently. Knowledge of a base and strong generating set essentially reduces these tasks to the calculation of certain orbits.

Given an arbitrary generating set for a permutation group, the Schreier-Sims algorithm calculates a base and strong generating set. We describe several variations of this method, including the Todd-Coxeter, random and extending Schreier-Sims algorithms.

Matrix groups over finite fields can also be represented by a base and strong generating set, by considering their action on the underlying vector space. A practical implementation of the random Schreier-Sims algorithm for matrix groups is described. We investigate the effectiveness of this implementation for computing with soluble groups, almost simple groups, simple groups and related constructions.

We consider in detail several aspects of the implementation of the random Schreier-Sims algorithm. In particular, we examine the generation of random group elements and choice of "stopping condition". The main difficulty in applying this algorithm to matrix groups is that the orbits which must be calculated are often very large. Shorter orbits can be found by extending the point set to include certain subspaces of the underlying vector space. We demonstrate that even greater improvements in the performance of the random Schreier-Sims algorithm can be achieved by using the orbits of eigenvectors and eigenspaces of the generators of the group.

Contents

Chapter 1

Background and notation

1.1 An overview

The design of algorithms for computing with finite permutation groups has been a particularly successful area of research within computational group theory. Such algorithms played an important part in the classification of the finite simple groups and have applications in other areas of mathematics, such as graph theory and combinatorics. An introduction to permutation group algorithms can be found in Butler (1991) and a recent survey of computational group theory is provided by Cannon & Havas (1992).

Successful computing with a permutation group is largely dependent on our ability to find an effective representation for the group. In particular, many calculations can be facilitated if we have a descending chain of subgroups, together with a set of coset representatives for each subgroup of the chain in its predecessor. We can construct a chain in which each subgroup is a point stabiliser of the last. The sequence of points stabilised is called a base, and a set containing generators for each subgroup in the chain is called a strong generating set; these concepts were introduced by Sims (1971) as an effective description of a permutation group. Sets of coset representatives for this chain can be constructed by calculating orbits of the base points. In Chapter 2, we define these concepts more formally and describe their fundamental applications, which include calculating the order of a group and testing for group membership. Our construction of the sets of coset representatives combines the new concept of a Schreier structure with ideas from earlier descriptions, such as Leon (1991) and Butler (1986).

Given a group described by generating permutations, the Schreier-Sims algorithm constructs a base and strong generating set for it. This algorithm was first described in Sims (1970) and uses Schreier's lemma to find generating sets for the stabilisers. However, these sets usually contain many redundant generators, so variations of the algorithm have been developed which reduce the number of generators which are considered. The Todd-Coxeter Schreier-Sims algorithm (Sims, 1978) uses coset enumeration to calculate the index of one stabiliser in its predecessor, and so determine if we have a complete generating set; coset enumeration is briefly described in Section 1.3. The random Schreier-Sims algorithm (Leon, 1980b) uses random elements, rather than Schreier generators, to construct a probable base and strong generating set. We can then use the Todd-Coxeter Schreier-Sims algorithm (among others) to verify that this probable base and strong generating set is complete. The extending Schreier-Sims algorithm (Butler, 1979) efficiently constructs a base and strong generating set for certain subgroups of interest. All of these algorithms are described in Chapter 3.

Many of the algorithms which have been developed for matrix groups over finite fields are modifications of permutation group algorithms. In particular, we can represent a matrix group by a base and strong generating set, if we consider its natural action on the underlying vector space. In Chapter 4, we describe the theory behind such modifications and discuss Schreier structures for matrix groups. The algorithms in Chapter 3 were first implemented for matrix groups by Butler (1976; 1979). As part of this project a new implementation of the random Schreier-Sims algorithm has been developed. We investigate the effectiveness of this implementation with soluble groups, almost simple groups, simple groups and related constructions.

In Chapter 5, we consider issues related to the implementation of the random Schreier-Sims algorithm. First, we study generating random elements of a group, a fundamental task in computational group theory. We briefly discuss methods suggested in the literature, and compare the impact of different generation schemes on our implementation. Next, we discuss conditions which are used to terminate the random Schreier-Sims algorithm. The three most common stopping conditions are compared in detail for the first time.

The main difficulty with using the random Schreier-Sims algorithm for matrix groups is that the base points often have very large orbits. In the final two sections we describe methods for finding base points with smaller orbits. Butler (1976) considered the action of matrix groups on sets other than the underlying vector space; in particular, he used the set of one-dimensional subspaces. We consider some other point sets and reject them on several grounds, before investigating Butler's method in detail. We find that his technique is effective for soluble groups, but large orbits must still be calculated for most simple groups. Finally, we discuss new methods for choosing base points which we expect *a priori* to have smaller orbits. In particular, we consider choosing the following as the first base point: an eigenvector of one generating matrix, a eigenvector common to two generators, or an eigenspace of a generator. These lead to significant improvements in the efficiency of the implementation from both space and time criteria.

Much recent progress has been made with matrix group computation in the context of the "recognition project", which is discussed in Neumann & Praeger (1992). This project seeks to develop algorithms which, given a matrix group, recognise its category in Aschbacher's 1984 classification of the subgroups of the general linear group. It is expected that the category of almost simple groups will pose the greatest difficulty. Following a suggestion by Praeger, we have investigated whether the random Schreier-Sims algorithm is particularly effective for certain almost simple groups. With most of these groups, we found that Butler's method worked well. However, in a few cases, we found extremely large orbits, even if the first base point was taken to be an eigenvector. We handle these groups by taking several base points to be carefully chosen eigenvectors.

These new techniques extend significantly the range of application of the random Schreier-Sims algorithm for matrix groups over finite fields. Similar methods should also be useful with the other variations of the Schreier-Sims algorithm.

1.2 Notation

Most of our notation is from Suzuki (1982), for general group theory, and Wielandt (1964), for permutation groups. An index of notation and definitions is provided. Let G be a *group*. The *product* of $g, h \in G$ is written $g \cdot h$ and the *identity* of G is denoted by e. We consider only finite groups. The number of elements in G is called the *order* and is written $|G|$. The *trivial* group is denoted by 1. We refer to simple groups using the notation of the Atlas (Conway, Curtis, Norton, Parker & Wilson, 1985).

If H is a *subgroup* of G, we write $H \leq G$. The subgroup *generated* by the set X is written $\langle X \rangle$ and the subgroup generated by $X \cup \{g\}$ is written $\langle X, g \rangle$. A (*right*) *coset* of H in G is a set of the form

$$H \cdot g = \{h \cdot g : h \in H\},$$

for some $g \in G$. The *index* of H in G is $|G : H| = |G| \, / \, |H|$. A set of *coset representatives* for H in G contains exactly one element from each coset, including the identity representing H itself.

Let Ω be an arbitrary finite set, we call its elements *points*. The number of points in a subset Σ of Ω is denoted by $|\Sigma|$ and called the *length*. A *permutation* on Ω is a one-to-one mapping from Ω onto itself. The *image* of $\alpha \in \Omega$ under the action of the permutation g is denoted by α^g. If $\alpha^g = \alpha$, we say that α is *fixed* by g. The *product* of the permutations g and h is defined by $\alpha^{g \cdot h} = (\alpha^g)^h$ for all $\alpha \in \Omega$. With this product, the set S_Ω of all permutations on Ω is a group, called the *symmetric group* on Ω. If G is a subgroup of S_Ω, then the *degree* of G is the length of Ω. Since Ω is finite, we can assume that Ω is just $\{1, 2, \ldots, n\}$, and write S_n for S_Ω.

The *disjoint union* of a family of sets is considered to be the normal union, under the assumption that the sets are pairwise disjoint. We denote the disjoint union of the sets A and B by $A \,\dot{\cup}\, B$.

A *directed, labelled graph* $\mathcal{G}$ is a triple (V, L, E) where V is the set of *vertices*, L is the set of *labels* and $E \subseteq V \times V \times L$ is the set of *edges*. If $(\beta, \gamma, l) \in E$, then $\mathcal{G}$ contains an edge from β to γ labelled by l; this is drawn as

$$\beta \xrightarrow{l} \gamma.$$

If $\mathcal{G}' = (V, L, E')$ with $E' \subseteq E$, we say that $\mathcal{G}'$ is a *subgraph* of $\mathcal{G}$. The *restriction* of $\mathcal{G}$ to $V' \subseteq V$ is the graph $(V', L, E \cap (V' \times V' \times L))$. A *path* from β to γ is a finite sequence of edges in $\mathcal{G}$ of the form

$$\beta = \alpha_1 \xrightarrow{l_1} \alpha_2 \xrightarrow{l_2} \cdots \xrightarrow{l_{m-1}} \alpha_m \xrightarrow{l_m} \alpha_{m+1} = \gamma;$$

the *length* of this path is m. The *distance* from β to γ is the length of the shortest path from β to γ. We consider the empty sequence to be a path of length zero from β to β.

We present algorithms in pseudo-code. Most statements are in English with mathematical notation, but the control structures are similar to those of Pascal. In particular, we use the following Pascal commands: **procedure** and **function** calls, **for** and **while** loops, and **if-then-else** statements. The statement **var** indicates an argument whose value can be changed by the procedure or function. We use the command **return** to exit from some procedures and all functions. Assignment is denoted by the command **let**; for example, the statement

let $i = i + 1$

increments the value of i by one. Unless otherwise stated, a set is stored as a list, and so has a linear ordering. A loop of the form

for $x \in X$ **do**

considers all of the elements of the set X in this order. If an element is added to X during the execution of such a loop, it is added to the end of the list and so will be considered in its turn; this is particularly important for the orbit algorithms in Section 2.3. Nested **for** loops are abbreviated as

for $a \in A$, **for** $b \in B$ **do**
$\vdots$
end for.

Our implementation of the random Schreier-Sims algorithm was written in traditional C (Kernighan & Ritchie, 1988). Most of the (representations of) matrix groups described in Section 4.4 are from the libraries of the computational algebra systems Cayley (Cannon, 1984) and GAP (Schönert *et al.*, 1993), or were constructed using GAP. These systems were also used for many other computations.

1.3 Coset enumeration

The Todd-Coxeter algorithm, or coset enumeration, is a method for finding the index of a subgroup in a finitely presented group. It was first described by

Todd & Coxeter (1936) as a method for hand calculation. In this section, we briefly describe the method, which is used in the Todd-Coxeter Schreier-Sims algorithm of Section 3.3. A detailed discussion of coset enumeration can be found in Neubüser (1982); our description is based on Sims (1978).

First we briefly discuss presentations for groups; see Johnson (1990) for more details. Let X be an arbitrary (finite) set, and let X^{-1} be another set with a one-to-one correspondence $x \leftrightarrow x^{-1}$ between X and X^{-1}. We denote $X \dot{\cup} X^{-1}$ by $X^{\pm 1}$. A *word* in X is a finite sequence of elements of $X^{\pm 1}$. We write words in the form

$$y_1 \cdot y_2 \cdot \cdots \cdot y_m,$$

where each y_i is in $X^{\pm 1}$. The set of all words in X is denoted $\mathcal{W}(X)$. The *product* of the words w_1 and w_2 is denoted by $w_1 \cdot w_2$ and the *inverse* of the word w is written w^{-1}. The *free group* on X is denoted by $F(X)$. If $\mathcal{R}$ is a subset of $\mathcal{W}(X)$, we define $N(\mathcal{R})$ to be the smallest normal subgroup containing all of the elements of $F(X)$ which correspond to words in $\mathcal{R}$.

Let G be a group with generating set X. The pair $\{X; \mathcal{R}\}$ is a *presentation* for G if there is an isomorphism from G to $F(X)/N(\mathcal{R})$ which takes x to $N(\mathcal{R}) \cdot x$ for all $x \in X$. If w is a word in X, we say that we *evaluate* w to find the corresponding element of G.

Suppose that $\{X; \mathcal{R}\}$ is a presentation for G. Let $\mathcal{S}$ be a set of words in X and let H be the subgroup of G whose generators are obtained by evaluating the words in $\mathcal{S}$. A (*partial*) *coset table* for H in G is a finite set Λ together with a function

$$t : \Lambda \times X^{\pm 1} \to \Lambda \dot{\cup} \{0\},$$

where the elements of Λ correspond to cosets of H in G. We denote by ι the element of Λ corresponding to H itself. The function t describes the action of the generators and their inverses on the cosets, where $t(\lambda, y) = 0$ indicates that the action of y on λ is unknown. We can extend t to a function $t' : \Lambda \times \mathcal{W}(X) \to \Lambda \dot{\cup} \{0\}$ by inductively defining

$$t'(\lambda, w \cdot y) = \begin{cases} t(t'(\lambda, w), y) & \text{if } t'(\lambda, w) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

for all $\lambda \in \Lambda$, $w \in \mathcal{W}(X)$ and $y \in X^{\pm 1}$.

In the remainder of the section let $\lambda \in \Lambda$, $y \in X^{\pm 1}$ and $w_1, w_2 \in \mathcal{W}(X)$. All coset tables must have the following properties:

1. If $t(\lambda, y) \neq 0$, then $t(t(\lambda, y), y^{-1}) = \lambda$.

2. For all $\lambda \in \Lambda$, there is a $w \in \mathcal{W}(X)$ such that $\lambda = t'(\iota, w)$.

The elements of Λ correspond to cosets of H in G if they also satisfy:

3. If $w_1 \cdot y \cdot w_2 \in \mathcal{R}$, $\mu = t'(\lambda, w_1) \neq 0$ and $\nu = t'(\lambda, w_2^{-1}) \neq 0$, then $t(\mu, y) = \nu$.

4. If $w_1 \cdot y \cdot w_2 \in \mathcal{S}$, $\mu = t'(\iota, w_1) \neq 0$ and $\nu = t'(\iota, w_2^{-1}) \neq 0$, then $t(\mu, y) = \nu$.

We call a coset table *closed* if $t(\lambda, y) \neq 0$, for all $\lambda \in \Lambda$, $y \in X^{\pm 1}$. A closed coset table for H in G faithfully represents the action of the elements of X on the cosets of H. In particular, the index of H in G is the number of elements in Λ.

The Todd-Coxeter algorithm constructs a coset table. It is essentially a systematic way of applying the following three operations to a coset table:

1. Define a new coset:
   If $t(\lambda, y) = 0$, then we can add a new coset μ to Λ and set $t(\lambda, y) = \mu$.

2. Make a deduction:
   Suppose $w_1 \cdot y \cdot w_2 \in \mathcal{R}$, or $w_1 \cdot y \cdot w_2 \in \mathcal{S}$ and $\lambda = \iota$. If $\mu = t'(\lambda, w_1) \neq 0$, $\nu = t'(\lambda, w_2^{-1}) \neq 0$ and $t(\mu, y) = t(\nu, y^{-1}) = 0$, then we can set $t(\mu, y) = \nu$ and $t(\nu, y^{-1}) = \mu$.

3. Force an equivalence:
   Suppose $w_1 \cdot w_2 \in \mathcal{R}$, or $w_1 \cdot w_2 \in \mathcal{S}$ and $\lambda = \iota$. If $\mu = t'(\lambda, w_1) \neq 0$, $\nu = t'(\lambda, w_2^{-1}) \neq 0$ and $\mu \neq \nu$, then μ and ν represent the same coset. Let $\sim$ be the smallest equivalence relation on Λ such that $\mu \sim \nu$ and, for all $y \in X^{\pm 1}$, $t(\lambda_1, y) \sim t(\lambda_2, y)$ whenever $\lambda_1 \sim \lambda_2$. Then we can replace Λ by a set of representatives of the equivalence classes of $\sim$, and replace every value of t by its representative.

Since the index of H in G is not necessarily finite, the repeated application of these operations need not result in a closed coset table. We ensure that the algorithm terminates by choosing a positive integer M, and not defining any new cosets after the size of Λ reaches M.

There are a number of variations of the Todd-Coxeter method which differ mainly in the order in which they apply these operations and the methods they use to handle equivalences. A recent discussion of these variations can be found in Havas (1991).

Chapter 2

Fundamental concepts

If we wish to investigate the structure of a group using a computer, we must be able to represent it by some data structure. If our group is finite, this data structure should assist us to perform the following basic tasks:

- Find the order of the group.
- List the group elements without repetition.
- Generate random group elements.
- Test for membership of the group.
- Store group elements efficiently.

These computations play an important role in many investigations of group structure. Perhaps the most natural representation for a permutation or matrix group is a generating set. In this chapter, we describe another representation which allows us to perform these computations more efficiently.

2.1 Chains of subgroups

The concept of a chain of subgroups is important in the representation of groups on a computer. Let G be a finite group.

Definition 2.1.1 *A* chain of subgroups *of G is a sequence of the form*

$$G = G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(k)} \geq G^{(k+1)} = 1.$$

If we have a chain of subgroups, then, for $i = 1, 2, \ldots, k$, we can choose a set $U^{(i)}$ consisting of coset representatives for $G^{(i+1)}$ in $G^{(i)}$. An element g of G is contained in exactly one coset of $G^{(2)}$ in $G^{(1)}$, so $g = h \cdot u_1$ for some unique h in $G^{(2)}$ and u_1 in $U^{(1)}$. By induction, we can show that

$$g = u_k \cdot u_{k-1} \cdot \cdots \cdot u_1$$

where each $u_i \in U^{(i)}$ is uniquely determined by g. The ability to write every group element uniquely as a product of this form underpins many of the applications of chains of subgroups in computational group theory.

A chain of subgroups of G, with corresponding sets of coset representatives, helps us to perform the tasks listed above. The order of G is simply the product of the sizes of the sets of coset representatives. We can list the elements of the group without repetition by evaluating all the words of the form $u_k \cdot u_{k-1} \cdot \cdots \cdot u_1$. A random element of the group can be generated by taking a random element from each $U^{(i)}$ and multiplying them in the appropriate sequence. If we can find some method for either writing a permutation in the form $u_k \cdot u_{k-1} \cdot \cdots \cdot u_1$, or proving it cannot be written in this form, then we have a membership test for G. Finally, we can store group elements as words of this form, but this is only useful if we have a memory efficient representation for the elements of the coset representative sets.

Chains of subgroups are also important in other branches of group theory; for example, a composition series is a way of exhibiting the structure of a group. Other chains include the derived series, and the upper and lower central series. In addition, other algebraic structures can usefully be described in terms of chains of substructures. For example, if V is a vector space with basis $[b_1, b_2, \ldots, b_k]$, then we have a chain of subspaces

$$V = V^{(1)} \geq V^{(2)} \geq \cdots \geq V^{(k)} \geq V^{(k+1)} = 0,$$

where $V^{(i)} = \langle b_i, b_{i+1}, \ldots, b_k \rangle$.

In the next section, we define a particularly useful chain of subgroups of a permutation group. In the subsequent sections of this chapter we consider in more detail how to use this chain to carry out the tasks listed at the beginning of the chapter.

2.2 Bases and strong generating sets

The concept of a base and strong generating set was introduced by Sims (1971). It provides a concise description of a particular chain of subgroups of a permutation group and is the basis of many permutation group algorithms. In Chapter 3, we discuss algorithms which construct a base and strong generating set for a group described by a generating set.

Let G be a permutation group on Ω. We can use the action of G on Ω to define certain subgroups.

Definition 2.2.1 *For β in Ω, the* stabiliser *of β in G is*

$$G_\beta = \{g \in G : \beta^g = \beta\}.$$

Clearly this is a subgroup. Given $\beta_1, \beta_2, \ldots, \beta_i \in \Omega$, we can inductively define

$$G_{\beta_1,\beta_2,\ldots,\beta_i} = \left(G_{\beta_1,\beta_2,\ldots,\beta_{i-1}}\right)_{\beta_i} = \{g \in G : \beta_j^g = \beta_j, \text{ for } j = 1, 2, \ldots, i\}.$$

We now define the concept of a base, and its associated chain of subgroups.

Definition 2.2.2 *A* base *for G is a finite sequence $B = [\beta_1, \beta_2, \ldots, \beta_k]$ of distinct points in Ω such that*

$$G_{\beta_1,\beta_2,\ldots,\beta_k} = 1.$$

Hence, the only element of G which fixes all of the points $\beta_1, \beta_2, \ldots, \beta_k$ is the identity. Clearly every permutation group has a base, but not all bases for a given group are of the same length. If we write $G^{(i)} = G_{\beta_1,\beta_2,\ldots,\beta_{i-1}}$, then we have a *chain of stabilisers*

$$G = G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(k)} \geq G^{(k+1)} = 1.$$

We often require that a base has the additional property that $G^{(i)} \neq G^{(i+1)}$.

It is useful to have a generating set for every subgroup in our chain.

Definition 2.2.3 *A* strong generating set *for G with respect to B is a set S of group elements such that, for $i = 1, 2, \ldots, k$,*

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

Note that $S \cap G^{(i)}$ is just the set of elements in S which fix $\beta_1, \beta_2, \ldots, \beta_{i-1}$. We write $S^{(i)}$ for $S \cap G^{(i)}$. Since each strong generating set is associated with a base it is useful to consider the two as a single object, which we often refer to by the abbreviation BSGS.

We now give some examples of bases and strong generating sets.

Example 2.2.1 The symmetric group S_n has a base $[1, 2, \ldots, n-1]$ and strong generating set

$$\{(1,2), (2,3), \ldots, (n-1, n)\}\,;$$

the alternating group A_n has a base $[1, 2, \ldots, n-2]$ and strong generating set

$$\{(1,2,3), (2,3,4), \ldots, (n-2, n-1, n)\}\,.$$

In fact, S_n has no base with fewer than $n-1$ points and A_n has no base with fewer than $n-2$ points. A base whose length is almost the degree of the group is of little use for computation. However, many interesting groups have a base which is short relative to the length of Ω; these include all of the simple groups except the alternating groups.

Example 2.2.2 The dihedral group on n points has a base $[1, 2]$ and strong generating set $\{(1, 2, \ldots, n),\ (2, n)(3, n-1)\ldots\}$.

Example 2.2.3 The Mathieu group on 11 points, M_{11}, has a base $[1, 2, 3, 4]$ and strong generating set $\{s_1, s_2, \ldots, s_7\}$, where:

$$\begin{aligned}
s_1 &= (1,10)(2,8)(3,11)(5,7),\\
s_2 &= (1,4,7,6)(2,11,10,9),\\
s_3 &= (2,3)(4,5)(6,11)(8,9),\\
s_4 &= (3,5,7,9)(4,8,11,6),\\
s_5 &= (4,6)(5,11)(7,10)(8,9),\\
s_6 &= (4,10,6,7)(5,9,11,8),\\
s_7 &= (4,11,6,5)(7,8,10,9).
\end{aligned}$$

Note that s_1 and s_2 suffice to generate M_{11}.

2.3 Orbits and Schreier structures

We now have a concise description for a chain of stabilisers. We also want sets of coset representatives for the chain, as we saw in Section 2.1. In general, such sets can be very difficult to construct, but for stabilisers they can be found relatively easily.

The concept of an orbit is used in the construction of sets of coset representatives for stabilisers. Let G be a permutation group on the point set Ω.

Definition 2.3.1 *For $\beta \in \Omega$, the* orbit *of β under the action of G is*

$$\beta^G = \{\beta^g : g \in G\}.$$

The following theorem gives us a one-to-one correspondence between the orbit of a point and the set of cosets of its point stabiliser.

Theorem 2.3.1 *Let G be a permutation group on Ω and let $\beta \in \Omega$. If $\gamma \in \beta^G$, then $\{g \in G : \beta^g = \gamma\}$ is a coset of G_β.*

Proof:
Choose $h \in G$ such that $\beta^h = \gamma$; then

$$\begin{aligned} \{g \in G : \beta^g = \gamma\} &= \left\{g \in G : \beta^g = \beta^h\right\} \\ &= \left\{g \in G : \beta^{g \cdot h^{-1}} = \beta\right\} \\ &= \left\{g \in G : g \cdot h^{-1} \in G_\beta\right\} = G_\beta \cdot h. \end{aligned}$$ □

From this theorem it follows that the length of the orbit of β is the same as the index of G_β in G; that is, $\left|\beta^G\right| = |G : G_\beta|$. Another consequence is that if we have a function $u : \beta^G \to G$ such that $\beta^{u(\gamma)} = \gamma$ and $u(\beta) = e$, then its image is a set of coset representatives for G_β. We call such a function a *representative function*. Note that the representative of the coset $G_\beta \cdot h$ is just $u(\beta^h)$.

Suppose G has a base $B = [\beta_1, \beta_2, \ldots, \beta_k]$.

Definition 2.3.2 *For $i = 1, 2, \ldots, k$, the* ith basic orbit, *denoted by $\Delta^{(i)}$, is ${\beta_i}^{G^{(i)}}$, and the* ith basic index *is the length of the* ith *basic orbit.*

The main purpose of a strong generating set is to allow us to calculate the basic orbits. By Theorem 2.3.1, there is a one-to-one correspondence between the ith basic orbit and the set of cosets of $G^{(i+1)}$ in $G^{(i)}$. In particular,

$$\left|\Delta^{(i)}\right| = \left|G^{(i)} : G^{(i+1)}\right|,$$

so the order of G is simply the product of the basic indices. In addition, we can store the sets of coset representatives for our chain as *basic representative functions* $u_i : \Delta^{(i)} \to G^{(i)}$ such that $\beta_i^{u_i(\gamma)} = \gamma$ and $u_i(\beta_i) = e$. We use the term *Schreier structure* to refer to a data structure which can be used to store the basic representative functions. We describe two Schreier structures in this section. In Section 4.2, we give analogous Schreier structures for matrix groups. It is often convenient to write $\mathbf{\Delta} = \left[\Delta^{(1)}, \Delta^{(2)}, \ldots, \Delta^{(k)}\right]$ and $\boldsymbol{u} = [u_1, u_2, \ldots, u_k]$.

Recall our assumption that Ω is $\{1, 2, \ldots, n\}$. An obvious Schreier structure is the $k \times n$ array with values in $G \dot\cup \{0\}$ defined by

$$\mathcal{U}(i, \gamma) = \begin{cases} u_i(\gamma) & \text{for } \gamma \in \Delta^{(i)} \\ 0 & \text{otherwise} \end{cases}.$$

Since this array occupies as much memory as n^2k integers, it is impractical for groups whose degree is even moderately large.

We now present a Schreier structure which enables us to recalculate $u_i(\gamma)$ as a word in the strong generators whenever we need it. This structure requires considerably less memory than the one given above, at the cost of taking more time to find the coset representatives. First we need a new way of viewing the action of the generators of a group on an orbit. Suppose G has generating set $X = \{x_1, x_2, \ldots, x_m\}$ and let $\Delta \subseteq \Omega$ be an orbit of G. Let $\mathcal{G}$ be a directed, labelled graph with vertex set Δ, label set $\{1, 2, \ldots, m\}$ and edges

$$\gamma \xrightarrow{i} \beta \text{ where } \gamma^{x_i} = \beta.$$

We say that $\mathcal{G}$ *represents the action* of X on Δ.

Example 2.3.1 Consider $M_{11} = \langle s_1, s_2 \rangle$, where s_1 and s_2 are defined in Example 2.2.3. The action of $\{s_1, s_2\}$ on the orbit $\Omega = \{1, 2, \ldots, 11\}$ is shown in Figure 2.1, where the dotted lines are labelled by 1 and solid lines by 2.

Suppose $\mathcal{G}$ represents the action of X on β^G, for some $\beta \in \Omega$. If we have a path in $\mathcal{G}$ of the form

$$\beta = \alpha_1 \xrightarrow{j_1} \alpha_2 \xrightarrow{j_2} \cdots \xrightarrow{j_{l-1}} \alpha_l \xrightarrow{j_l} \alpha_{l+1} = \gamma,$$

then $\beta^{x_{j_1} \cdot x_{j_2} \cdot \cdots \cdot x_{j_l}} = \gamma$. Suppose that, for every point $\gamma \in \beta^G$, we can choose a unique path in $\mathcal{G}$ from β to γ; then we have a unique element which takes β to γ, expressed as a word in the generators. This would give us a representative function for the cosets of G_β in G.

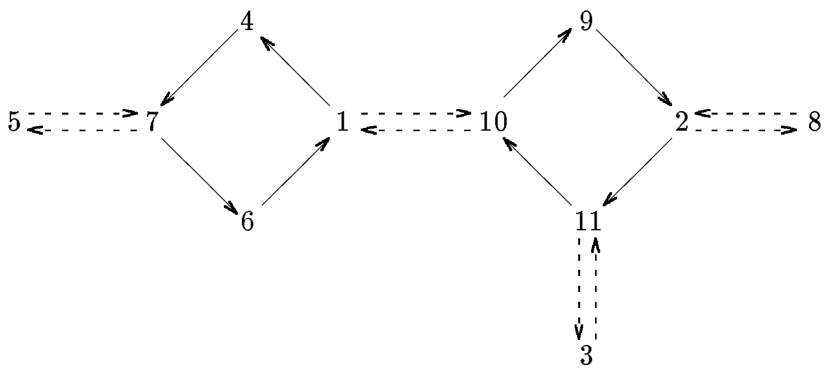

Figure 2.1: Graph representing the action of the generators of M_{11}

Definition 2.3.3 *For a graph $\mathcal{G} = (V, L, E)$ and $\beta \in V$, a* spanning tree *for $\mathcal{G}$ rooted at β is a subgraph $\mathcal{T}$ which contains, for every $\gamma \in V$, a unique path in $\mathcal{T}$ from β to γ.*

An arbitrary directed, labelled graph need not have a spanning tree, and it may have spanning trees rooted at some vertices but not others. However, if the graph represents the action of a generating set on an orbit, it has a spanning tree rooted at every point in that orbit.

Theorem 2.3.2 *Let X be a generating set for G and let $\mathcal{G}$ be the graph representing the action of X on an orbit Δ of G. If $\beta \in \Delta$, then $\mathcal{G}$ has a spanning tree rooted at β.*

Proof: Since every element of G has finite order, we can write the inverse of an element as a power, so every element of G is a product of elements of X. Hence, there is at least one path from β to any point in Δ. We can now construct our spanning tree by induction on the distance from β. Suppose we have a spanning tree $\mathcal{T}$ for the restriction of $\mathcal{G}$ to the set of points of distance less than l from β. Then, for each $\gamma \in \Delta$ of distance l from β, choose a path

$$\beta = \alpha_1 \xrightarrow{j_1} \alpha_2 \xrightarrow{j_2} \cdots \xrightarrow{j_{l-1}} \alpha_l \xrightarrow{j_l} \alpha_{l+1} = \gamma$$

of length l. Let $\mathcal{T}'$ be the graph which contains all of the edges of $\mathcal{T}$ as well as the edges

$$\alpha_l \xrightarrow{j_l} \gamma$$

from the paths chosen above. It is easily shown that $\mathcal{T}'$ is a spanning tree for the restriction of $\mathcal{G}$ to the set of points of distance less than $l+1$ from β. $\Box$

The following theorem helps us to construct an efficient data structure for a spanning tree.

Theorem 2.3.3 *If $\mathcal{T}$ is a spanning tree for $\mathcal{G} = (V, L, E)$ rooted at β, then, for every $\gamma \in V \setminus \{\beta\}$, there is a unique edge in $\mathcal{T}$ ending at γ.*

Proof: Suppose $\mathcal{T}$ contains the edges

$$\alpha \xrightarrow{l} \gamma \xleftarrow{l'} \alpha'$$

where $(\alpha, l) \neq (\alpha', l')$. Then the unique paths in $\mathcal{T}$ from β to α and from β to α' can be extended by these edges to give two different paths from β to γ. $\Box$

If we know the unique edge in $\mathcal{T}$ ending at γ for every $\gamma \in V \setminus \{\beta\}$, then clearly we have the entire spanning tree. Hence, $\mathcal{T}$ can be represented by functions $v : V \to L \dot{\cup} \{-1\}$ and $\omega : V \to V \dot{\cup} \{-1\}$ defined by:

$$v(\gamma) = \begin{cases} l & \text{if } \alpha \xrightarrow{l} \gamma \text{ is in } \mathcal{T} \\ -1 & \text{if } \gamma = \beta \end{cases},$$

$$\omega(\gamma) = \begin{cases} \alpha & \text{if } \alpha \xrightarrow{l} \gamma \text{ is in } \mathcal{T} \\ -1 & \text{if } \gamma = \beta \end{cases}.$$

The pair (v, ω) is called the *linearised version* of $\mathcal{T}$. When $\mathcal{T}$ is a spanning tree for a graph representing the action of a generating set on an orbit, we call v the *Schreier vector* and ω the *vector of backward pointers.*

Finally, we are able to describe our second Schreier structure. Suppose G has a base $B = [\beta_1, \beta_2, \ldots, \beta_k]$ and strong generating set $S = \{s_1, s_2, \ldots, s_m\}$. For $i = 1, 2, \ldots, k$, let $\mathcal{G}_i$ be the graph representing the action of $S^{(i)}$ on $\Delta^{(i)}$. By Theorem 2.3.2, there is a spanning tree $\mathcal{T}_i$ for $\mathcal{G}_i$ rooted at β_i. This gives us a basic representative function u_i, since, for every $\gamma \in \Delta^{(i)}$, the tree $\mathcal{T}_i$ contains a unique path

$$\beta_i = \alpha_1 \xrightarrow{j_1} \alpha_2 \xrightarrow{j_2} \cdots \xrightarrow{j_{l-1}} \alpha_l \xrightarrow{j_l} \alpha_{l+1} = \gamma,$$

so we can define $u_i(\gamma) = s_{j_1} \cdot s_{j_2} \cdot \cdots \cdot s_{j_l}$. Let (v_i, ω_i) be the linearised version of the tree $\mathcal{T}_i$. Then our Schreier structure is simply this pair of functions stored as the $k \times n$ array with values in $\mathbb{Z} \times \mathbb{Z}$ defined by

$$\mathcal{V}(i, \gamma) = \begin{cases} (v_i(\gamma), \omega_i(\gamma)) & \text{for } \gamma \in \Delta^{(i)} \\ (0, 0) & \text{otherwise} \end{cases} .$$

Example 2.3.2 Consider M_{11} with the base and strong generating set of Example 2.2.3. Then $\mathcal{G}_1$ contains the graph shown in Figure 2.1, where dotted lines are labelled by 1 and solid lines by 2. A spanning tree $\mathcal{T}_1$ rooted at 1 is shown in Figure 2.2. Schreier vectors and vectors of backward pointers for this base and strong generating set are given in Table 2.1.

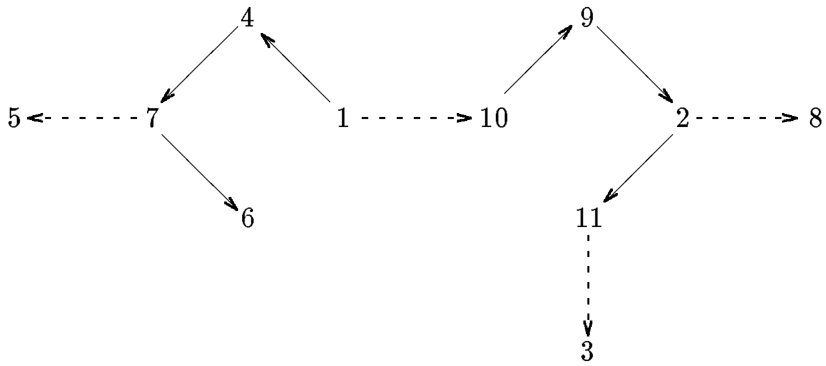

Figure 2.2: Spanning tree $\mathcal{T}_1$ for M_{11}

	1	2	3	4	5	6	7	8	9	10	11
v_1	-1	2	1	2	1	2	2	1	2	1	2
ω_1	-1	9	11	1	7	7	4	2	10	1	2
v_2	0	-1	3	7	4	7	4	6	4	6	5
ω_2	0	-1	2	5	3	11	5	11	7	4	5
v_3	0	0	-1	7	4	7	4	6	4	6	5
ω_3	0	0	-1	5	3	11	5	11	7	4	5
v_4	0	0	0	-1	7	5	6	6	7	6	7
ω_4	0	0	0	-1	6	4	6	11	10	4	4

Table 2.1: Schreier structure $\mathcal{V}$ for M_{11}

We use the following algorithm to calculate the value of a representative function from a Schreier vector and a vector of backward pointers.

Algorithm 2.3.1 *Trace a spanning tree*

function *trace*(γ, X, v, ω)
(* input: generating set $X = \{x_1, x_2, \ldots, x_m\}$,
linearised spanning tree (v, ω) for the orbit containing γ.
output: return $u = u(\gamma)$. *)
begin
    let $u = e$**,** $\alpha = \gamma$**;**
    while $v(\alpha) \neq -1$ **do**
        let $u = x_{v(\alpha)} \cdot u$**;**
        let $\alpha = \omega(\alpha)$**;**
    end while;
    return u**;**
end function.

In Section 2.5, we need to calculate the inverses of the coset representatives. This can easily be achieved by replacing $x_{v(\alpha)} \cdot u$ by $u \cdot x_{v(\alpha)}^{-1}$ in the algorithm above. We can facilitate this calculation by using (strong) generating sets which are closed under inversion. In addition, this makes backward pointers unnecessary, since, for any $\alpha \in \beta^G \setminus \{\beta\}$,

$$\omega(\alpha) = \alpha^{x_j^{-1}},$$

where $j = v(\alpha)$.

Note that in both of the Schreier structures described above we have critically used the natural linear ordering on Ω. In Chapter 4, we discuss Schreier structures for matrix groups where no such natural ordering exists. Another Schreier structure is the *labelled branching tree*, which is used in the complexity analysis of the Schreier-Sims algorithm, as explained in Butler (1991).

2.4 Computing orbits

We now describe an algorithm for calculating the orbit of a point. Let G be a permutation group on Ω with generating set X, and let β be a point in Ω. In the following algorithm, we denote the set of points currently known to be in β^G by Δ and initialise it to be $\{\beta\}$. We then proceed to close Δ under the action of X. A linearised spanning tree for the graph representing the action of X on the orbit is also calculated.

Algorithm 2.4.1 *Calculate orbit*

> **function** *calculate_orbit*(β, X)
> (* input: $\beta \in \Omega$, generating set $X = \{x_1, x_2, \ldots, x_m\}$.
> output: $\Delta = \beta^G$, Schreier vector v, vector of backward pointers ω. *)
> **begin**
> **let** $\Delta = \{\beta\}$;
> **let** $v(\gamma) = \omega(\gamma) = \begin{cases} -1 & \text{for } \gamma = \beta \\ 0 & \text{otherwise} \end{cases}$ **;**
> **for** $\delta \in \Delta$, **for** $x_j \in X$ **do**
> **let** $\gamma = \delta^{x_j}$;
> **if** $\gamma \notin \Delta$ **then**
> **add** γ **to** Δ;
> **let** $v(\gamma) = j$, $\omega(\gamma) = \delta$;
> **end if;**
> **end for;**
> **return** Δ, v, ω;
> **end function.**

This algorithm terminates after it has applied every generator to every point in Δ. Normally, we store (v, ω) as an array of length n with values in $\mathbb{Z} \times \mathbb{Z}$, where an entry $(0, 0)$ in position γ indicates that $\gamma \notin \Delta$. Hence, we can easily check if γ is in Δ, and we no longer need Δ after the algorithm terminates.

Frequently, we wish to calculate $\beta^{\langle X \rangle}$ when we already know $\beta^{\langle \widetilde{X} \rangle}$ for some $\widetilde{X} \subseteq X$. The following variation of our orbit algorithm does this without recalculating the entire orbit. We have omitted the calculation of the Schreier vector and vector of backward pointers, but this can easily be added.

Algorithm 2.4.2 *Extend orbit*

```
function extend_orbit(β, X, X̃, Δ̃)
(* input:  β ∈ Ω, X̃ ⊆ X ⊆ S_Ω, Δ̃ = β^⟨X̃⟩.
   output: Δ = β^⟨X⟩. *)
begin
    let Δ = Δ̃;
    for δ ∈ Δ, for x ∈ X do
        if δ ∉ Δ̃ or x ∉ X̃ then
            let γ = δ^x;
            if γ ∉ Δ then
                add γ to Δ;
            end if;
        end if;
    end for;
    return Δ;
end function.
```

2.5 Testing group membership

We now consider a method for testing whether a given permutation is an element of a group. Let G be a permutation group on Ω with base $B = [\beta_1, \beta_2, \ldots \beta_k]$ and strong generating set S. Suppose that we also know the basic orbits and basic representative functions with respect to this base. An arbitrary permutation g on Ω is an element of G if and only if we can express it in the form

$$g = u_k(\gamma_k) \cdot u_{k-1}(\gamma_{k-1}) \cdot \cdots \cdot u_1(\gamma_1),$$

where every $\gamma_i \in \Delta^{(i)}$.

Theorem 2.5.1 *Let G be a permutation group on Ω and let $\beta \in \Omega$. Let $u : \beta^G \to G$ be a representative function. If $g \in G$ then, for some $h \in G_\beta$,*

$$g = h \cdot u\left(\beta^g\right).$$

Proof:
We have $\beta^g = \beta^{u(\beta^g)}$, so $\beta^{g \cdot u(\beta^g)^{-1}} = \beta$. Hence, $h = g \cdot u(\beta^g)^{-1} \in G_\beta$. □

This theorem gives us an algorithm for testing whether a permutation is an element of G. If g is a permutation on Ω with $\beta_1^g \notin \Delta^{(1)}$, then we know that g is not in G. Otherwise, we can write $g = h \cdot u_1(\beta_1^g)$, and then $g \in G$ if and only if $h \in G^{(2)}$. The problem has now been reduced to testing whether h is in $G^{(2)}$. Iterating this process, we find that g can be written in the form

$$g = \bar{g} \cdot u_{l-1}(\gamma_{l-1}) \cdot u_{l-2}(\gamma_{l-2}) \cdot \cdots \cdot u_1(\gamma_1),$$

where $\beta_l^{\bar{g}} \notin \Delta^{(l)}$ or $l = k+1$. Clearly, g is in G if and only if $\bar{g} = e$. This algorithm, called *stripping* with respect to B, is presented more formally below. Notice that $\bar{g}$ and l are uniquely determined by g; we call $\bar{g}$ the *residue* and l the *drop-out level* from stripping g.

Algorithm 2.5.1 *Stripping permutations*

function $strip(g, B, \boldsymbol{\Delta}, \boldsymbol{u})$
(∗ input: $g \in G$, base $B = [\beta_1, \beta_2, \ldots, \beta_k]$ with corresponding $\boldsymbol{\Delta}$, $\boldsymbol{u}$.
output: residue $\bar{g}$ and drop-out level l. ∗)
begin
  let $\bar{g} = g$;
  for $l = 1$ **to** k **do**
    if $\beta_l^{\bar{g}} \notin \Delta^{(l)}$ **then**
      return $\bar{g}$, l;
    else
      let $\bar{g} = \bar{g} \cdot (u_l(\beta_l^{\bar{g}}))^{-1}$;
    end if;
  end for;
  return $\bar{g}$, $k+1$;
end function.

This algorithm also plays an important role in Chapter 3, where it is used to decide if a particular element of a group should be used as a strong generator.

2.6 Representing group elements

We now discuss representations for the elements of a group described by a base and strong generating set. These representations are more memory efficient than storing a permutation, and they help us to generate random group elements and list the elements without repetition.

Let G be a permutation group on Ω, with base $B = [\beta_1, \beta_2, \ldots, \beta_k]$. Suppose we have an element g of G. We can strip it and write it in the form

$$g = u_k(\gamma_k) \cdot u_{k-1}(\gamma_{k-1}) \cdot \cdots \cdot u_1(\gamma_1),$$

where each $\gamma_i \in \Delta^{(i)}$. The sequence $[\gamma_1, \gamma_2, \ldots, \gamma_k]$ could then be used to represent g. This is efficient in terms of memory usage, but two elements stored in this form must both be converted to permutations before their product can be calculated. Note that we can generate a random group element by choosing a random point γ_i from each $\Delta^{(i)}$ and calculating $u_k(\gamma_k) \cdot u_{k-1}(\gamma_{k-1}) \cdot \cdots \cdot u_1(\gamma_1)$.

We now present an alternative representation which has the same memory requirements but is easier to work with.

Definition 2.6.1 *The* base image *of $g \in G$ with respect to the base B is*

$$B^g = [\beta_1^g, \beta_2^g, \ldots, \beta_k^g] .$$

The following theorem shows that base images are indeed a faithful representation of the group elements.

Theorem 2.6.1 *If the permutation group G has a base B, then the function $g \mapsto B^g$ is one-to-one on G.*

Proof:
Suppose g and h are elements of G with $B^g = B^h$, then

$$\left[\beta_1^{g \cdot h^{-1}}, \beta_2^{g \cdot h^{-1}}, \ldots, \beta_k^{g \cdot h^{-1}}\right] = [\beta_1, \beta_2, \ldots, \beta_k] .$$

Hence $g \cdot h^{-1} \in G_{\beta_1, \beta_2, \ldots, \beta_k} = 1$, and so $g = h$. □

If g and h are in G and $[\alpha_1, \alpha_2, \ldots, \alpha_k]$ is the base image of g, then the base image of $g \cdot h$ is simply $\left[\alpha_1^h, \alpha_2^h, \ldots, \alpha_k^h\right]$. Hence, to multiply two elements stored as base images, it is only necessary to convert *one* of them into a permutation. It is easy to convert a permutation to its base image; the following algorithm performs the converse.

Algorithm 2.6.1 *Convert from base image to permutation*

function *base_image_to_permutation*$(A, \boldsymbol{u})$
(* input: base image $A = [\alpha_1, \alpha_2, \ldots, \alpha_k]$, basic representative functions $\boldsymbol{u}$.
output: the group element g corresponding to A. *)
begin
  let $g = e$;
  for $i = 1$ **to** k **do**
    let $g = u_i(\alpha_i) \cdot g$;
    for $j = i + 1$ **to** k **do**
      let $\alpha_j = \alpha_j^{u_i(\alpha_i)^{-1}}$;
    end for;
  end for;
  return g;
end function.

Finally, we consider the problem of listing all the elements of a group without repetition. We can easily enumerate all the sequences $[\gamma_1, \gamma_2, \ldots, \gamma_k]$ with each γ_i in $\Delta^{(i)}$ and calculate the products of the form

$$u_k(\gamma_k) \cdot u_{k-1}(\gamma_{k-1}) \cdot \cdots \cdot u_1(\gamma_1).$$

However, this involves many permutation multiplications, and so is time consuming. Often it suffices to enumerate all the base images of the elements of G. If $g = u_k(\gamma_k) \cdot u_{k-1}(\gamma_{k-1}) \cdot \cdots \cdot u_1(\gamma_1)$ is an element of G, then

$$\beta_i^g = \beta_i^{u_i(\gamma_i) \cdot u_{i-1}(\gamma_{i-1}) \cdot \cdots \cdot u_1(\gamma_1)};$$

this follows immediately from the fact that $u_j(\gamma_j) \in G^{(j)}$ fixes $\beta_1, \beta_2, \ldots, \beta_{j-1}$. Given our list of sequences $[\gamma_1, \gamma_2, \ldots, \gamma_k]$, we can use this observation to enumerate the base images. Note that this method involves calculating images of points rather than multiplying permutations.

The ability to list the base images allows us to carry out a *backtrack search*. This technique can be used to construct a BSGS for a subgroup of G consisting of all the elements satisfying a certain property. Examples of such subgroups include centralisers, normalisers, set stabilisers and intersections. A detailed discussion of this method can be found in Butler (1982). A backtrack search can often be facilitated by choosing a base with certain properties. Sims (1971) described an effective method for finding a strong generating set with respect to a given base, provided we already have some base and strong generating set for the group.

Chapter 3

The Schreier-Sims algorithm

In Chapter 2 we considered the use of a base and strong generating set as a computationally efficient representation for a permutation group. We now discuss several algorithms which construct such a representation for a group given by generating permutations. These are all variations of the Schreier-Sims algorithm described in Section 3.2.

3.1 Partial BSGS and Schreier's lemma

We use the following definitions and notation throughout this chapter. Let G be a permutation group on Ω with a generating set X. Suppose $B = [\beta_1, \beta_2, \ldots, \beta_k]$ is a sequence of points in Ω and S is a subset of G. We call B a *partial base* and S a *partial strong generating set* if S contains X and is closed under inversion, and no element of S fixes every point in B. An ordinary base and strong generating set is called *complete* if there is any chance of confusion. Note that k is used to denote the length of both partial and complete bases. Let $i = 1, 2, \ldots, k$ and define $G^{(i)} = G_{\beta_1,\beta_2,\ldots,\beta_{i-1}}$ and $S^{(i)} = S \cap G^{(i)}$. In addition, write $H^{(i)} = \left\langle S^{(i)} \right\rangle$ and $\Delta^{(i)} = \beta_i^{H^{(i)}}$. We now have

$$G = G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(k)} \geq G^{(k+1)},$$

$$G = H^{(1)} \geq H^{(2)} \geq \cdots \geq H^{(k)} \geq H^{(k+1)} = 1,$$

$$G^{(i+1)} = {G^{(i)}}_{\beta_i} \geq {H^{(i)}}_{\beta_i} \geq H^{(i+1)}.$$

A *partial basic representative function* $u_i : \Delta^{(i)} \to H^{(i)}$ has the property that $\beta_i^{u_i(\gamma)} = \gamma$ and $u_i(\beta_i) = e$. These functions can be represented by a Schreier structure as shown in Section 2.3, and $U^{(i)} = u_i(\Delta^{(i)})$ is a set of coset representatives for ${H^{(i)}}_{\beta_i}$ in $H^{(i)}$. Finally, we write $\boldsymbol{\Delta} = \left[\Delta^{(1)}, \Delta^{(2)}, \ldots, \Delta^{(k)}\right]$ and $\boldsymbol{u} = [u_1, u_2, \ldots, u_k]$.

If we have a subset S of G and a sequence B of points, we can use the following algorithm to extend them to a partial base and strong generating set. In particular, we can find a partial BSGS for a group given by a generating set by calling this procedure with $B = [\,]$ and $S = \emptyset$.

Algorithm 3.1.1 *Find partial BSGS*

```
procedure partial_BSGS(var B, var S, X)
(* input:  S ⊆ G, sequence B of points, generating set X.
   output: partial BSGS B and S. *)
begin
    let S = (S ∪ X)\{e};
    let T = S;
    for s ∈ T do
        if B^s = B then
            find a point β not fixed by s;
            add β to B;
        end if;
        if s^2 ≠ e then
            add s^-1 to S;
        end if;
    end for;
end procedure.
```

If $\Omega = \{1, 2, \ldots, n\}$, then a point not fixed by a permutation is found by simply considering each point in turn. With matrix groups, we use the method described in Section 4.3.

The following theorem, presented in Leon (1980b), is used to verify the correctness of several of the algorithms in this chapter.

Theorem 3.1.1 *Suppose G has a partial base B and partial strong generating set S. Then the following are equivalent:*

(i) *B is a base and S is a strong generating set.*

(ii) $H^{(i+1)} = G^{(i+1)}$, *for* $i = 1, 2, \ldots, k$.

(iii) ${H^{(i)}}_{\beta_i} = H^{(i+1)}$, *for* $i = 1, 2, \ldots, k$.

(iv) $\left|H^{(i)} : H^{(i+1)}\right| = \left|\Delta^{(i)}\right|$, *for* $i = 1, 2, \ldots, k$.

Proof:
By definition, (i) is equivalent to (ii). From (ii) we can deduce

$$ {H^{(i)}}_{\beta_i} = {G^{(i)}}_{\beta_i} = G^{(i+1)} = H^{(i+1)}, $$

so we have (iii). Conversely, if (iii) is true and $G^{(j)} = H^{(j)}$, then

$$ G^{(j+1)} = {G^{(j)}}_{\beta_j} = {H^{(j)}}_{\beta_j} = H^{(j+1)}; $$

so, by induction on j, (iii) implies (ii). Finally, (iii) is equivalent to (iv) since $H^{(i+1)} \subseteq {H^{(i)}}_{\beta_i}$ and $\left|H^{(i)} : {H^{(i)}}_{\beta_i}\right| = \left|\Delta^{(i)}\right|$. □

A key component of the Schreier-Sims algorithm is Schreier's lemma, which allows us to write down a generating set for the stabiliser of a point. Our proof follows that of Hall (1959).

Theorem 3.1.2 *Suppose G is a group with generating set X, and H is a subgroup of G. If U is a set of coset representatives for H in G, and the function $t : G \to U$ maps an element g of G to the representative of $H \cdot g$, then a generating set for H is given by*

$$ \left\{u \cdot x \cdot t(u \cdot x)^{-1} : u \in U, x \in X\right\}. $$

Proof:
Every element h of H can be written in the form $y_1 \cdot y_2 \cdot \cdots \cdot y_l$, where each y_i, or its inverse, is in X. Let $u_i = t(y_1 \cdot y_2 \cdot \cdots \cdot y_i)$, for $i = 0, 1, \ldots, l$. Then $u_0 = t(e) = e$ and $u_l = t(h) = e$, so

$$ h = u_0 \cdot h \cdot u_l^{-1} = (u_0 \cdot y_1 \cdot u_1^{-1}) \cdot (u_1 \cdot y_2 \cdot u_2^{-1}) \cdot \cdots \cdot (u_{l-1} \cdot y_l \cdot u_l^{-1}). $$

Consider $u_{i-1} \cdot y_i \cdot u_i$, for $i = 1, 2, \ldots, l$. Now $u_i = t(y_1 \cdot y_2 \cdot \cdots \cdot y_i) = t(u_{i-1} \cdot y_i)$, since $H \cdot y_1 \cdot y_2 \cdot \cdots \cdot y_i = H \cdot u_{i-1} \cdot y_i$. Let $u = u_{i-1} \in U$ and $y = y_i$; we can now write

$$u_{i-1} \cdot y_i \cdot u_i^{-1} = u \cdot y \cdot t(u \cdot y)^{-1}.$$

This has the desired form if $y \in X$; otherwise, let $y = x^{-1}$ for some $x \in X$ and let $v = t(u \cdot x^{-1}) \in U$. Since $H \cdot v \cdot x = H \cdot u \cdot x^{-1} \cdot x$, we have $t(v \cdot x) = u$, and so the inverse of $u \cdot y \cdot t(u \cdot y)^{-1}$ can be written

$$t(u \cdot x^{-1}) \cdot x \cdot u^{-1} = v \cdot x \cdot t(v \cdot x)^{-1},$$

which has the desired form. The result now follows. □

The generators of H given by this theorem are called *Schreier generators.*

Suppose that G is a permutation group on Ω and $H = G_\beta$ for some $\beta \in \Omega$. Let $u : \beta^G \to G$ be a representative function for G_β in G. If we take our set of coset representatives to be $U = u(\Delta)$, then, by Theorem 2.3.1, $t(g) = u(\beta^g)$, for all g in G. Hence

$$t(u(\alpha) \cdot x) = u(\beta^{u(\alpha) \cdot x}) = u(\alpha^x),$$

for all $\alpha \in \beta^G$, and so the set of Schreier generators for G_β is

$$\left\{ u(\alpha) \cdot x \cdot u(\alpha^x)^{-1} : \alpha \in \beta^G, x \in X \right\}.$$

This immediately suggests an algorithm for finding a base and strong generating set. If G has a partial base $B = [\beta_1, \beta_2, \ldots, \beta_k]$ and a partial strong generating set S, then we can find Schreier generators for each $G^{(i+1)}$ by induction on i. When the algorithm terminates, $H^{(i+1)} = G^{(i+1)}$ for $i = 1, 2, \ldots, k$. Hence, if we ensure that B, S remains a partial BSGS, then, by Theorem 3.1.1, we have a complete BSGS. This is presented more formally as Algorithm 3.1.2.

In Hall (1959), it is shown that $|G : H| - 1$ of the Schreier generators for H are the identity; so the number of non-trivial generators is

$$1 + |G : H| \left(|X| - 1 \right),$$

if we ignore the possibility of repetitions among them. Hence, by induction, the set of Schreier generators for $G^{(i)}$ could be as large as

$$1 + \left|\Delta^{(1)}\right| \cdot \left|\Delta^{(2)}\right| \cdots \left|\Delta^{(i)}\right| \left(|X| - 1 \right).$$

In the next section we consider another algorithm based on Schreier's lemma, which constructs a much smaller strong generating set.

Algorithm 3.1.2 *Application of Schreier's Lemma*

> **begin**
> **let** X **be a generating set of** G**;**
> **find a partial base** B **and strong generating set** S**;**
> **for** $\beta_i \in B$ **do**
> $Schreier(B, S, i)$**;**
> **end for;**
> **end.**
>
> **procedure** $Schreier($**var** $B,$ **var** $S, i)$
> (* input: partial BSGS $B = [\beta_1, \beta_2, \ldots, \beta_k]$ and S
>   such that $G^{(j+1)} = H^{(j+1)}$ for $j = 1, 2, \ldots, i-1$.
> output: extended partial BSGS $B = [\beta_1, \beta_2, \ldots, \beta_{k'}]$ and S
>   such that $G^{(j+1)} = H^{(j+1)}$ for $j = 1, 2, \ldots, i$. *)
> **begin**
> **calculate** $\Delta^{(i)}$ **and** u_i**;**
> **let** $T = S^{(i)}$**;**
> **for** $\alpha \in \Delta^{(i)}$**, for** $s \in T$ **do**
>   **let** $g = u_i(\alpha) \cdot s \cdot u_i(\alpha^s)^{-1}$**;**
>   **if** $g \neq e$ **then**
>   **add** g **and** g^{-1} **to** S**;**
>   **if** B **is fixed by** g **then**
>   **add to** B **some point not fixed by** g**;**
>   **end if;**
>   **end if;**
> **end for;**
> **end procedure.**

3.2 The Schreier-Sims algorithm

The Schreier-Sims algorithm was first described in Sims (1970), where he discusses computational methods for determining primitive permutation groups. This description did not explicitly define the concept of a base and strong generating set; instead the base was always taken to be $[1, 2, \ldots, n]$ and generators for each stabiliser were considered separately, rather than as a single strong generating set.

Suppose we have a partial base B and partial strong generating set S, which we wish to extend to a complete BSGS. Before we add a new Schreier generator to S, we would like to test whether it is already redundant. We say that level i of the subgroup chain is *complete* if every Schreier generator of ${H^{(i)}}_{\beta_i}$ has been included in S or shown to be redundant. Recall that the Schreier generators of ${H^{(i)}}_{\beta_i}$ are elements of the form

$$u_i(\alpha) \cdot s \cdot u_i(\alpha^s)^{-1},$$

for $\alpha \in \Delta^{(i)} = \beta_i^{H^{(i)}}$ and $s \in S^{(i)}$. If level i is complete, then

$${H^{(i)}}_{\beta_i} = H^{(i+1)};$$

so, by Theorem 3.1.1, if every level of the chain is complete, we have a complete base and strong generating set.

The algorithm in the previous section completes the levels of the chain from the lowest to the highest. By proceeding from the highest level down, the Schreier-Sims algorithm provides us with a way of testing potential strong generators for redundancy. Suppose that we have completed levels $i+1, i+2, \ldots, k$, where k is the current length of B, then we know that

$${H^{(j)}}_{\beta_j} = H^{(j+1)}$$

for $j = i+1, i+2, \ldots, k$. Hence, $H^{(i+1)}$ has a base $[\beta_{i+1}, \beta_{i+2}, \ldots, \beta_k]$ and strong generating set $S^{(i+1)}$, by Theorem 3.1.1. We can now use the stripping algorithm described in Section 2.5 to test for membership of $H^{(i+1)}$.

Suppose g is a new Schreier generator for ${H^{(i)}}_{\beta_i}$. If g is in $H^{(i+1)}$, then it is redundant as a strong generator, so we need not add it to S. If g is not in

$H^{(i+1)}$, then the stripping algorithm returns a residue $\bar{g} \neq e$ and a drop-out level $l \geq i+1$. It is clear from the definition of a residue that

$$\left\langle H^{(i+1)}, g \right\rangle = \left\langle H^{(i+1)}, \bar{g} \right\rangle.$$

However, adding $\bar{g}$ to S also extends the groups $H^{(i+2)}, H^{(i+3)}, \ldots, H^{(l-1)}$, while adding g may not. Once $\bar{g}$ has been added to S, the levels $i+1, i+2, \ldots, l$ are no longer complete, so we continue our computation at level l. If $l = k+1$, then $\bar{g}$ fixes every point in B, so we must add a point not fixed by $\bar{g}$ to B; we can now continue at the newly created level $k+1$.

In summary, the Schreier-Sims algorithm starts at the highest level of the chain and attempts to find a Schreier generator whose residue is not the identity. If successful, it adds the residue to S, and continues at the drop-out level, after extending B if necessary. Otherwise, it continues at the next lowest level. This algorithm terminates because there can only be finitely many Schreier generators for each level of the chain. Algorithm 3.2.1 is a more formal description of it as a recursive procedure.

We describe one further improvement to the Schreier-Sims algorithm before we discuss some variations of it. Each time the procedure *Schreier-Sims1* is called at a given level, every possible Schreier generator is stripped, even if it has been checked during a previous call. The procedure *Schreier-Sims2*, presented as Algorithm 3.2.2, checks each Schreier generator only once. This is achieved by using the set $\widetilde{S}$, which contains the elements of S which were present the last time the procedure was called at the current level. We can also use $\widetilde{S}$ to calculate the basic orbits with the Extend orbit algorithm given in Section 2.4. Note that S and $\widetilde{S}$ need not be separate sets, it is only necessary to record which strong generators have been added since the last call. We can calculate a BSGS with this procedure by initialising $\boldsymbol{\Delta}$ and $\boldsymbol{u}$ to be empty, and replacing the call to *Schreier-Sims1* in the main part of Algorithm 3.2.1 by

$$\textit{Schreier-Sims2}(B, S, \emptyset, \boldsymbol{\Delta}, \boldsymbol{u}, i).$$

Algorithm 3.2.1 *Schreier-Sims, version one*

begin
  let X **be a generating set for** G**;**
  find a partial base B **and strong generating set** S**;**
  for $i = k$ **down to** 1 **do**
    Schreier-Sims1(B, S, i)**;**
  end for;
end.

procedure *Schreier-Sims1*(**var** B, **var** S, i)
(∗ input: partial BSGS $B = [\beta_1, \beta_2, \ldots, \beta_k]$ and S
  such that ${H^{(j)}}_{\beta_j} = H^{(j+1)}$ for $j = i+1, i+2, \ldots, k$.
 output: extended partial BSGS $B = [\beta_1, \beta_2, \ldots, \beta_{k'}]$ and S
  such that ${H^{(j)}}_{\beta_j} = H^{(j+1)}$ for $j = i, i+1, \ldots, k'$. ∗)
begin
  calculate $\Delta^{(i)}$ **and** u_i**;**
  let $T = S^{(i)}$**;**
  for $\alpha \in \Delta^{(i)}$**, for** $s \in T$ **do**
    let $g = u_i(\alpha) \cdot s \cdot u_i(\alpha^s)^{-1}$**;**
    strip g **wrt** $H^{(i+1)}$ **to find residue** $\bar{g}$ **and drop-out level** l**;**
    if $\bar{g} \neq e$ **then**
      add $\bar{g}$ **and** $\bar{g}^{-1}$ **to** S**;**
      if $l = k+1$ **then**
        add to B **a point not fixed by** $\bar{g}$**;**
      end if;
      for $j = l$ **down to** $i+1$ **do**
        Schreier-Sims1(B, S, j)**;**
      end for;
    end if;
  end for;
end procedure.

Algorithm 3.2.2 *Schreier-Sims, version two*

procedure *Schreier-Sims2*(**var** B, **var** $S, \widetilde{S}$, **var** $\boldsymbol{\Delta}$, **var** $\boldsymbol{u}, i$)
(∗ input: partial BSGS $B = [\beta_1, \beta_2, \ldots, \beta_k]$ and S
such that ${H^{(j)}}_{\beta_j} = H^{(j+1)}$ for $j = i+1, i+2, \ldots, k$,
old strong generating set $\widetilde{S}$ with corresponding $\boldsymbol{\Delta}$ and $\boldsymbol{u}$.
output: extended partial BSGS $B = [\beta_1, \beta_2, \ldots, \beta_{k'}]$ and S
such that ${H^{(j)}}_{\beta_j} = H^{(j+1)}$ for $j = i, i+1, \ldots, k'$. ∗)
begin
    let $\widetilde{\Delta} = \Delta^{(i)}$;
    extend $\Delta^{(i)}$ **and** u_i;
    let $T = S^{(i)}$;
    for $\alpha \in \Delta^{(i)}$, **for** $s \in T$ **do**
        if $\alpha \notin \widetilde{\Delta}$ **or** $s \notin \widetilde{S}$ **then**
            let $g = u_i(\alpha) \cdot s \cdot u_i(\alpha^s)^{-1}$;
            strip g **wrt** $H^{(i+1)}$ **to find residue** $\bar{g}$ **and drop-out level** l;
            if $\bar{g} \neq e$ **then**
                add $\bar{g}$ **and** $\bar{g}^{-1}$ **to** S;
                if $l = k+1$ **then**
                    add to B **a point not fixed by** $\bar{g}$;
                end if;
                for $j = l$ **down to** $i+1$ **do**
                    Schreier-Sims2$(B, S, S \setminus \{\bar{g}, \bar{g}^{-1}\}, \boldsymbol{\Delta}, i)$;
                end for;
            end if;
        end if;
    end for;
end procedure.

3.3 The Todd-Coxeter Schreier-Sims algorithm

The most time consuming part of the Schreier-Sims algorithm is stripping the Schreier generators, because this involves a large number of group multiplications. We now present a method, based on coset enumeration, which verifies that we have completed a given level of the chain, without having to strip every Schreier generator.

At the ith level, the Schreier-Sims algorithm ensures that ${H^{(i)}}_{\beta_i} = H^{(i+1)}$, by showing that all the Schreier generators of ${H^{(i)}}_{\beta_i}$ are in $H^{(i+1)}$. However, by Theorem 3.1.1, it suffices to show that

$$\left|H^{(i)} : H^{(i+1)}\right| = \left|\Delta^{(i)}\right|.$$

If we have a presentation for $H^{(i)}$, then we can calculate this index by the technique of coset enumeration described in Section 1.3. Relators for $H^{(i)}$ can be obtained from the process of stripping the Schreier generators. If g is a Schreier generator for ${H^{(i)}}_{\beta_i}$, then stripping it with respect to $H^{(i+1)}$ provides us with a relation of the form

$$g = \bar{g} \cdot u_{l-1}(\gamma_{l-1}) \cdot u_{l-2}(\gamma_{l-2}) \cdot \cdots \cdot u_{i+1}(\gamma_{i+1}),$$

where l is the drop-out level and $\bar{g}$ is the residue. If we use the Schreier structure $\mathcal{V}$ described in Section 2.3, then we know each $u_j(\gamma_j)$ as a word in $S^{(j)} \subseteq S^{(i)}$. In addition, g is of the form $u_i(\alpha) \cdot s \cdot u_i(\alpha^s)^{-1}$, for some $\alpha \in \Delta^{(i)}$ and $s \in S^{(i)}$, so it is also known as a word in $S^{(i)}$. Hence, if $\bar{g} = e$, we have a new relator

$$g \cdot u_{i+1}(\gamma_{i+1})^{-1} \cdot \cdots \cdot u_{l-2}(\gamma_{l-2})^{-1} \cdot u_{l-1}(\gamma_{l-1})^{-1}$$

in the existing elements of $S^{(i)}$; otherwise we add $\bar{g}$ to S (and, thus, to $S^{(i)}$), and we have a relator

$$g \cdot u_{i+1}(\gamma_{i+1})^{-1} \cdot \cdots \cdot u_{l-2}(\gamma_{l-2})^{-1} \cdot u_{l-1}(\gamma_{l-1})^{-1} \cdot \bar{g}^{-1}$$

in the elements of $S^{(i)}$. We denote the set of all relators obtained in this way by $\mathcal{R}$, and we write $\mathcal{R}^{(i)}$ for the set of relators in $\mathcal{R}$ which only involve elements of $S^{(i)}$. Often we find that $\{S^{(i)}; \mathcal{R}^{(i)}\}$ is a presentation for $H^{(i)}$ after only a few Schreier generators have been stripped at level i. In addition to finding a base and strong generating set, the Todd-Coxeter Schreier-Sims algorithm constructs a presentation $\{S; \mathcal{R}\}$ for our group.

Algorithm 3.3.1 *Todd-Coxeter Schreier-Sims*

```
procedure Todd-Coxeter_Schreier-Sims(var B, var S, var R, c, i)
(* input:  partial BSGS B = [β_1, β_2, ..., β_k] and S
           such that |H^(i) : H^(i+1)| = |Δ^(i)| for j = i+1, i+2, ..., k,
           set R of relators, real number c ≥ 1.
   output: extended partial BSGS B = [β_1, β_2, ..., β_k'] and S
           such that |H^(i) : H^(i+1)| = |Δ^(i)| for j = i, i+1, ..., k'. *)
begin
    calculate Δ^(i) and u_i;
    let T = S^(i);
    for α ∈ Δ^(i), for s ∈ T do
        if |Δ^(i)| = Todd-Coxeter(S^(i), R^(i), S^(i+1), c . |Δ^(i)|) then
            return;
        end if;
        let g = u_i(α) · s · u_i(α^s)^-1;
        strip g wrt H^(i+1) to find residue ḡ and drop-out level l;
        if ḡ = e then
            add g · u_{i+1}(γ_{i+1})^-1 · ... · u_{l-2}(γ_{l-2})^-1 · u_{l-1}(γ_{l-1})^-1 to R;
        else
            add ḡ and ḡ^-1 to S;
            add g · u_{i+1}(γ_{i+1})^-1 · ... · u_{l-2}(γ_{l-2})^-1 · u_{l-1}(γ_{l-1})^-1 · ḡ^-1 to R;
            if l = k + 1 then
                add to B a point not fixed by ḡ;
            end if;
            for j = l down to i + 1 do
                Todd-Coxeter_Schreier-Sims(B, S, R, c, j);
            end for;
        end if;
    end for;
end procedure.
```

Algorithm 3.3.1 plays the same role as the procedure *Schreier-Sims1*. We have not incorporated the improvements of *Schreier-Sims2*, but these could easily be added. The function *Todd-Coxeter*$(X, \mathcal{R}, \mathcal{S}, M)$ enumerates cosets, as described in Section 1.3, until either the coset table closes or the number of cosets exceeds M. It returns the final size of the coset table. The real number c determines how large we let our coset tables become before we try to find a new relator, it is normally taken to be about 1.1.

The performance of the Todd-Coxeter Schreier-Sims algorithm is largely dependent on the particular method of coset enumeration used, as is discussed in Leon (1980b). Rather than recalculating our coset tables, we store one coset table for each level of the chain which we modify each time *Todd-Coxeter* is called at that level. This technique is called *interruptible* coset enumeration. Since we store a number of coset tables, the Todd-Coxeter Schreier-Sims algorithm uses more memory than the Schreier-Sims algorithm. This may make the algorithm impractical for groups of large degree.

3.4 The random Schreier-Sims algorithm

The random Schreier-Sims algorithm was first described by Leon (1980b) for use with the Todd-Coxeter Schreier-Sims algorithm (see Section 3.6). The use of Schreier generators generally results in an unnecessarily large strong generating set. The random Schreier-Sims algorithm avoids this problem by using random elements of the group; we discuss the generation of random group elements in Section 5.1. This method usually produces a complete BSGS very rapidly, but we have no way of knowing when it is complete. We decide to terminate the algorithm when some predetermined condition becomes true. We want this *stopping condition* to become true within a reasonable amount of time, while maximising our chances of finding a complete BSGS. The most common stopping conditions are discussed in Section 5.2.

During the execution of the random Schreier-Sims algorithm, we cannot be sure that we have a base and strong generating set for some non-trivial subgroup of G. However, if we apply the stripping algorithm given in Section 2.5 with respect to a partial BSGS, it tests for membership of the set

$$U^{(k)} \cdot U^{(k-1)} \cdot \ldots \cdot U^{(1)},$$

where $U^{(i)} = u_i(\Delta^{(i)})$ is a set of coset representatives for ${H^{(i)}}_{\beta_i}$ in $H^{(i)}$. Clearly, if a random element is in this set, then it is redundant as a strong generator, otherwise we add the residue of this stripping process to S.

Algorithm 3.4.1 *Random Schreier-Sims*

> **begin**
>   **let** X **be a generating set of** G**;**
>   **find a partial base** B **and strong generating set** S**;**
>   **while** *stopping_condition* = false **do**
>   **let** g **be a random element of** G**;**
>   **let** $\bar{g}$ **be the residue of stripping** g **wrt** B **and** S**;**
>   **if** $\bar{g} \neq e$ **then**
>   **add** $\bar{g}$ **and** $\bar{g}^{-1}$ **to** S**;**
>   **if** $B^{\bar{g}} = B$ **then**
>   **add to** B **a point not fixed by** $\bar{g}$**;**
>   **end if;**
>   **end if;**
>   **end while;**
> **end.**

In Section 3.6, we briefly discuss methods for verifying that the probable base and strong generating set produced by this algorithm is complete.

3.5 The extending Schreier-Sims algorithm

The variations of the Schreier-Sims algorithm described so far calculate a BSGS for a group given by a generating set. We now describe the extending Schreier-Sims algorithm, which was first implemented by Butler (1979), following a suggestion by Richardson. Assume we have a permutation group H on Ω, and a permutation g which is not in H. Suppose we also have a base B and strong generating set S for H. We wish to extend B and S to form a base and strong generating set for $\langle H, g\rangle$. Our algorithm uses the procedure *Schreier-Sims2* to perform this calculation efficiently.

Algorithm 3.5.1 *Extending Schreier-Sims*

begin

  let B **and** S **be a BSGS for** H**;**

  calculate $\boldsymbol{\Delta}$ **and** $\boldsymbol{u}$**;**

  let g **be a permutation not in** H**;**

  strip g **wrt** H **to find residue** $\bar{g}$ **and drop-out level** l**;**

  add $\bar{g}$ **and** $\bar{g}^{-1}$ **to** S**;**

  if $l = k + 1$ **then**

    add to B **a point not fixed by** $\bar{g}$**;**

  end if;

  for $i = l$ **down to** 1 **do**

    Schreier-Sims2$(B, S, S \backslash \{\bar{g}, \bar{g}^{-1}\}, \boldsymbol{\Delta}, \boldsymbol{u}, i)$**;**

  end for;

end.

The applications of this algorithm include computing normal closures, commutator subgroups, derived series, and lower and upper central series (see Butler & Cannon (1982)). In all of these applications, we are working entirely within a larger group G; that is, $H \leq G$ and $g \in G$. Frequently, we also know a base C for G, and so the elements of G can be represented as base images rather than permutations. This is most useful for stripping Schreier generators.

Algorithm 3.5.2 strips an element of G with respect to the base B for the subgroup H. It represents the element as an image of the base C. This is faster than the stripping algorithm in Section 2.5, because a base image can be calculated more rapidly than a product of two permutations. The residue is returned as a base image A, and also as a word in $S \cup \{g\}$ (since we have each $u_i(\alpha)$ as a word in S). If $A = C$, then the residue is trivial; otherwise we can calculate it as a permutation by evaluating the word. Most of the Schreier generators which are stripped by the extending Schreier-Sims algorithm have trivial residues, so this method can lead to significant improvements in the efficiency of the algorithm. The known base stripping algorithm can also be used with the other variations of the Schreier-Sims algorithm, if we already know a base for the group which we are considering. The random Schreier-Sims algorithm is particularly effective when a BSGS is known, because we can also use it to generate random elements, as described in Section 2.6.

Algorithm 3.5.2 *Known base stripping*

function $strip(g, B, \mathbf{\Delta}, \boldsymbol{u}, C)$
(* input: $g \in G$, base B with corresponding $\mathbf{\Delta}$ and $\boldsymbol{u}$
for the group H, base C for G.
output: residue $\bar{g}$ as a word and base image A, drop-out level l *)
begin
let $A = C^g$**;**
let $\bar{g} = g$ **as a word;**
for $l = 1$ **to** k **do**
let $\alpha = \beta_l^{\bar{g}}$**;**
if $\alpha \notin \Delta^{(l)}$ **then**
return $\bar{g}$**,** A**,** l**;**
else
let $\bar{g} = \bar{g} \cdot (u_l(\alpha))^{-1}$ **as a word;**
let $A = A^{u_l(\alpha)^{-1}}$**;**
end if;
end for;
return $\bar{g}$**,** A**,** $k+1$**;**
end function.

3.6 Choice of algorithm

We conclude this chapter with a brief discussion on which variation of the Schreier-Sims algorithm is most effective for a particular group; a more detailed discussion can be found in Cannon & Havas (1992). If we wish to find a BSGS for a group of degree under 100, the standard Schreier-Sims algorithm is generally the fastest method. If the degree exceeds about 300, it is more efficient to construct a probable BSGS with the random Schreier-Sims algorithm, and then apply an algorithm to *verify* that it is complete, or show that it is not. The Todd-Coxeter Schreier-Sims algorithm is an effective verification algorithm for groups having degrees in the low thousands, and it completes the probable

BSGS if necessary. If our group has higher degree, the Brownie-Cannon-Sims verification algorithm, which is discussed in Bosma & Cannon (1992), is generally more efficient. In recent years significant progress has been made in the construction of bases and strong generating sets for groups of degree up to about 10 million. This is achieved by modifying these algorithms to use less memory; see, for example, Linton (1989).

If we know more about the group than just a generating set, we can often construct a BSGS more efficiently. One example of this is the extending Schreier-Sims algorithm. Another is described in Section 5.2, where we see that if the order of the group is known in advance, then the random Schreier-Sims algorithm can be used to construct a complete BSGS.

Chapter 4

The random Schreier-Sims algorithm for matrices

The algorithms described in Chapter 3 construct a base and strong generating set for a permutation group, which can then be used to investigate the group structure. We wish to make similar investigations into the structure of matrix groups over finite fields. Many of the algorithms which have been developed for matrix group computations are modifications of permutation group algorithms.

In this chapter, we first describe the theory required to adapt permutation group algorithms to work for matrix groups. Next, we consider data structures for orbits and representative functions of matrix groups. Finally, we describe an implementation of the random Schreier-Sims algorithm and discuss the matrix groups which were used to investigate the performance of this implementation. In Chapter 5 we consider further implementation issues, and report on some new methods for improving the range of application of the algorithm for matrix groups.

4.1 Conversion to matrix groups

Let $q = p^m$ be a power of a prime p and let $GF(q)$ be the unique field with q elements. The set of invertible $n \times n$ matrices over $GF(q)$ is denoted by $GL(n, q)$ and forms a group under matrix multiplication. The following definition is fundamental to the modification of permutation group algorithms for subgroups of $GL(n, q)$.

Definition 4.1.1 *Let G be a group and let S_Ω be the group of all permutations on an arbitrary finite set Ω. An* action *of G on Ω is a homomorphism from G to S_Ω.*

A group action is *faithful* if it is one-to-one. A faithful action of G on Ω is an isomorphism between G and a subgroup of S_Ω.

There is, of course, a natural action of a matrix group $G \leq GL(n, q)$ on the underlying vector space. Let $V(n, q)$ be the n-dimensional space of row vectors over the field $GF(q)$. We define the action of $A \in G$ on $v \in V(n, q)$ by $v^A = vA$. This action is clearly faithful, so G can be considered to be a permutation group on $V(n, q)$.

We can now apply permutation group algorithms to matrix groups. However, the number of vectors in $V(n, q)$ is q^n, which grows exponentially with n. This means that the basic orbits can be very large; with simple groups in particular, the first basic index is often the order of the group. This is demonstrated by the results in Table 4.1, where the first base point is a vector chosen by the algorithm outlined in Section 4.3. If we wish to investigate matrix groups using the random Schreier-Sims algorithm, we must find base points with smaller orbits.

Group	n	q	$\lvert\Delta^{(1)}\rvert$	$\lvert G^{(2)}\rvert$
A_8	28	11	20160	1
A_8	64	11	20160	1
M_{11}	55	7	3960	2
M_{11}	44	2	7920	1
M_{12}	120	17	95040	1
M_{22}	21	7	443520	1
M_{22}	34	2	18480	48
J_1	7	11	14630	12
J_1	27	11	87780	2
J_2	42	3	604800	2
$2.J_2$	36	3	1209600	1
$2.J_2$	42	3	60480	2
$4.J_2$	12	3	403200	6
$U_4(2)$	58	5	25920	1
$A_5 \times A_5$	25	7	3600	1

Table 4.1: Lengths of first basic indices

One technique for finding smaller basic orbits is to consider the action of G on a set other than $V(n,q)$; for example, the set of subspaces of $V(n,q)$. Often this means using an action which is not faithful, in which case we also take base points in $V(n,q)$, or else we only construct a BSGS for the quotient of G by the kernel of the action. In Section 5.3, we consider actions of matrix groups on several different sets. A second method, discussed in Section 5.4, reduces the orbit sizes by choosing points which we expect *a priori* to have small orbits.

4.2 Orbits and Schreier structures

In this section, we discuss the storage of basic orbits and representative functions for matrix groups. Suppose we have a group G with a base $[\beta_1, \beta_2, \ldots, \beta_k]$ and strong generating set S. Recall from Section 2.3 that we want a data structure for the basic orbits $\boldsymbol{\Delta} = \left[\Delta^{(1)}, \Delta^{(2)}, \ldots, \Delta^{(k)}\right]$, together with a Schreier structure for the basic representative functions $\boldsymbol{u} = [u_1, u_2, \ldots, u_k]$. With permutation groups on $\Omega = \{1, 2, \ldots, n\}$, a possible Schreier structure is the $k \times n$ array

$$\mathcal{U}(i,\gamma) = \begin{cases} u_i(\gamma) & \text{for } \gamma \in \Delta^{(i)} \\ 0 & \text{otherwise} \end{cases} .$$

An alternative is the $k \times n$ array

$$\mathcal{V}(i,\gamma) = \begin{cases} (v_i(\gamma), \omega_i(\gamma)) & \text{for } \gamma \in \Delta^{(i)} \\ (0,0) & \text{otherwise} \end{cases} ,$$

where (v_i, ω_i) is a linearised spanning tree of the graph representing the action of $S^{(i)}$ on $\Delta^{(i)}$. If we use either of these Schreier structures, we do not need to store the basic orbits separately, since $\alpha \in \Delta^{(i)}$ if and only if the (i, α) position of the array is non-zero.

These Schreier structures rely critically on the fact that the point set has a predetermined size and a natural linear ordering. If G is a matrix group, we can use points of many different types and it is impractical to impose an order on the entire point set. Instead, we choose a positive integer N, which we hope will be an upper bound on the lengths of the basic orbits of G. We can now store the basic orbits as a $k \times N$ array $\mathcal{O}$ where, for $i = 1, 2, \ldots, k$ and $m = 1, 2, \ldots, N$,

$$\mathcal{O}(i,m) \in \Delta^{(i)} \,\dot{\cup}\, \{0\};$$

and, for each $\alpha \in \Delta^{(i)}$, there is exactly one integer m such that $\mathcal{O}(i, m) = \alpha$. Now, for each $i = 1, 2, \ldots, k$, this array provides us with a one-to-one correspondence between $\Delta^{(i)}$ and some subset of $\{1, 2, \ldots, N\}$, so we can use the natural ordering on $\mathbb{N}$ to impose an ordering on each basic orbit.

We can now define Schreier structures for matrix groups which are analogous to $\mathcal{U}$ and $\mathcal{V}$. Our first Schreier structure is the $k \times N$ array with values in $G \dot\cup \{0\}$ defined by

$$\mathcal{U}'(i, m) = \begin{cases} u_i \circ \mathcal{O}(i, m) & \text{if } \mathcal{O}(i, m) \neq 0 \\ 0 & \text{otherwise} \end{cases} ;$$

however, this requires too much memory to be practical for most groups. Another, more memory efficient, Schreier structure is the $k \times N$ array with values in $\mathbb{Z} \times \mathbb{Z}$ given by

$$\mathcal{V}'(i, m) = \begin{cases} (v_i \circ \mathcal{O}(i, m), m') & \text{if } \mathcal{O}(i, m) \neq 0 \\ (0, 0) & \text{otherwise} \end{cases} ,$$

where m' is the unique integer such that $\mathcal{O}(i, m') = \omega_i \circ \mathcal{O}(i, m)$.

When the basic orbits and representative functions are stored in this way, we must make frequent searches for particular points in each row of the array $\mathcal{O}$. Probably the most efficient search method available is a *hash search* algorithm, several varieties of which are described in detail in Knuth (1973). Here we briefly describe the *linear probe* hash search algorithm. First, we must choose a *hash function*

$$h : \Omega \to \{1, 2, \ldots, N\},$$

where Ω contains all the different points we might use. If we wish to add α to $\Delta^{(i)}$, we store it as $\mathcal{O}(i, h(\alpha))$, unless that entry in the array already holds a point, in which case we store it in the next free position in the ith row of $\mathcal{O}$. We can now test a point α for membership of $\Delta^{(i)}$ by searching the ith row from position $h(\alpha)$, rather than from the beginning. This is an effective search algorithm, provided we choose a hash function which can be calculated rapidly and whose values are roughly uniformly distributed over the range $1, 2, \ldots, N$. If a row of the array $\mathcal{O}$ is almost full, the search algorithm slows down considerably; hence, N should be at least $4/3$ of the length of the longest basic orbit.

4.3 Implementations

The first matrix group implementation of the random Schreier-Sims algorithm was developed by Butler (1976; 1979), who also implemented the other algorithms described in Chapter 3. He chose his points to be one-dimensional subspaces and vectors, as described in Section 5.3. The basic orbits and representative functions were stored in data structures similar to $\mathcal{O}$ and $\mathcal{V}'$, and a hash search algorithm was used. He also carried out some preprocessing to find short orbits. This consisted of choosing a random point and building up its orbit until it was complete or it exceeded 5000 in length. If its orbit was complete, it was taken to be the first basic orbit, otherwise the process was repeated.

The random Schreier-Sims algorithm has been implemented for matrix groups over fields of prime order as part of this project. This implementation is written in traditional C (Kernighan & Ritchie, 1988). The orbits are stored as an array $\mathcal{O}$ which is searched with the linear probe hashing algorithm, and we use the Schreier structure $\mathcal{V}'$. We now present the algorithm used to calculate our hash function on the vectors in $V(n, q)$.

Algorithm 4.3.1 *Hash function*

function $h(v)$
(* input: vector $v = (v_1, v_2, \ldots, v_n)$.
  output: value of the hash function for v. *)
begin
  let $hash = 0$;
  for $i = 1$ **to** n **do**
    left shift $hash$;
    let $hash = hash$ xor v_i;
  end for;
  return $(hash \bmod N) + 1$;
end function.

We use the bitwise exclusive or operation (xor) on the entries of the vector because it is usually faster than arithmetical operations. A left shift by m bits, essentially multiplication by 2^m, is used to achieve a roughly uniform distribution of values for the hash function; the value of m used depends on the ratio of n to the number of bits used to store an entry in our vector. Similar hash functions can be defined for subspaces and other types of point.

Our implementation uses vectors and subspaces for the base points, as described in Section 5.3. It accepts from the user an arbitrary number of points, which are used as the initial base points. Whenever we find a new strong generator A which fixes the existing points, a new base point is chosen by the following algorithm. If A is not scalar, the vector chosen has the property that the one-dimensional subspace generated by it is also not fixed by A. Note that e_i is the ith standard basis vector; that is, the vector with one as its ith entry and zeros elsewhere.

Algorithm 4.3.2 *Choose new base point*

function *base_point*(A)
(* input: $I \neq A \in GL(n,q)$.
  output: $v \in V(n,q)$ such that $vA \neq v$ and, if possible, $vA \notin \langle v \rangle$. *)
begin
  for $i = 1$ **to** n, **for** $j = 1$ **to** n **do**
    if $i \neq j$ **and** $A_{ij} \neq 0$ **then** (* A not diagonal *)
      return e_i;
    end if;
  end for;
  for $i = 1$ **to** n, **for** $j = 1$ **to** n **do**
    if $i \neq j$ **and** $A_{ii} \neq A_{jj}$ **then** (* A not scalar *)
      return $e_i + e_j$;
    end if;
  end for;
  if $A_{11} \neq 1$ **then** (* $A \neq I$ *)
    return e_1;
  else
    return error;
  end if;
end function.

4.4 Evaluating performance

We used a variety of matrix groups to investigate the performance of our implementation of the random Schreier-Sims algorithm. These groups are of three types:

1. soluble groups;

2. almost simple groups;

3. simple groups and related constructions.

The results of these tests are given in the next chapter; for future reference, we list in this section all the groups mentioned there, together with their orders. If possible, groups are described using the notation of the Atlas (Conway *et al.*, 1985). Most of the groups were either obtained from the libraries of the computational algebra systems Cayley and GAP, or were constructed using GAP.

The soluble groups were constructed by calculating the wreath product of a soluble subgroup of a small general linear group and a p-group. The subgroups of general linear groups are from the libraries constructed by Short (1992), which contain all of the soluble subgroups of $GL(2,7)$, $GL(4,3)$, $GL(5,3)$ and $GL(6,2)$. The p-groups are subgroups of S_{p^m} for some m, constructed following the description in Hall (1959). In Table 4.2 we list the soluble groups used; the wreath product of the ith member of Short's $GL(n,q)$ library with a p-subgroup of S_{p^m} is denoted by $GL(n,q)$–i–p^m.

Group	n	q	Order
$GL(2,7)$–15–2^4	32	7	$2^{63}3^{16}$
$GL(2,7)$–3–3^3	54	7	$2^{81}3^{13}$
$GL(4,3)$–1–3^3	108	3	$3^{13}5^{27}$
$GL(4,3)$–1–2^4	64	3	$2^{15}5^{16}$
$GL(4,3)$–50–3^3	108	3	$2^{135}3^{40}$
$GL(5,3)$–2–3^3	135	3	$2^{27}3^{13}11^{27}$
$GL(5,3)$–9–2^4	80	3	$2^{95}5^{16}$
$GL(6,2)$–10–2^4	96	2	$2^{31}3^{48}$
$GL(6,2)$–21–3^3	162	2	$2^{27}3^{121}$
$GL(6,2)$–33–2^3	48	2	$2^{39}3^{24}$

Table 4.2: Soluble groups

Aschbacher's 1984 classification of the subgroups of $GL(n,q)$ divides them into nine categories. Various researchers are working on the matrix group "recognition project", which seeks to construct algorithms which recognise groups in these categories (see, for example, Neumann & Praeger (1992)). It is expected that the category of almost simple groups will pose the greatest difficulty.

Definition 4.4.1 *Suppose $G \leq GL(n,q)$ and Z is the subgroup of scalar matrices in G. Then G is* almost simple *if there is a non abelian simple group T such that $T \leq G/Z \leq \text{Aut } T$.*

In private communication, Praeger asked if the random Schreier-Sims algorithm is particularly effective when applied to almost simple groups. In particular, she suggested constructing the permutation module of S_m over $GF(q)$, and then applying the Meataxe to this reducible module. See Holt & Rees (1994) for a description of the Meataxe. The largest composition factor has dimension $m-1$, or $m-2$ if q divides m. This factor is isomorphic to S_m and is an almost simple group with $T = A_m$. The values of m and q which we used are listed in Table 4.3.

m	q	m	q	m	q
20	5	30	31	57	3
20	23	35	2	63	53
25	17	40	2	65	11
30	7	50	53	72	2

Table 4.3: Almost simple groups

Our final collection of test groups contains simple groups (mostly sporadic); covers, subgroups and other constructions based on simple groups; and also a couple of miscellaneous examples. The representations listed in Table 4.4 are obtained from the matrix group libraries of Cayley and GAP, and a number of other sources. Note that $(3^2{:}4 \times A_6)\cdot 2$ is a maximal subgroup of $3.O'N$; the group $GL(5,3)$ is represented as a subgroup of $GL(20,3)$; and $\phi F(2,9)$ is the representation of the quotient of the Fibonacci group, $F(2,9)$, constructed by Havas, Richardson & Sterling (1979).

Group	n	q	Order
A_8	20	11	20,160
A_8	28	11	20,160
A_8	64	11	20,160
M_{11}	24	3	7920
M_{11}	44	2	7920
M_{11}	44	7	7920
M_{11}	55	7	7920
M_{12}	55	7	95,040
M_{12}	120	17	95,040
M_{22}	21	7	443,520
$2.M_{22}$	34	2	887,040
J_1	7	11	175,560
J_1	14	11	175,560
J_1	27	11	175,560
$2.J_2$	36	3	1,209,600
$2.J_2$	42	3	1,209,600
$4.J_2$	12	3	2,419,200
$6.J_3$	18	2	301,397,760
$U_4(2)$	58	5	25,920
$U_4(2)$	64	2	25,920
$U_4(2)$	81	11	25,920
$(3^2{:}4\times A_6)\cdot 2$	18	7	77,760
$(3^2{:}4\times A_6)\cdot 2$	27	7	77,760
$(3^2{:}4\times A_6)\cdot 2$	45	7	77,760
$Sz(8)$	65	29	29,120
$4.Suz$	12	3	1,793,381,990,400
$A_5 \times A_5$	25	7	3600
$M_{11} \wr M_{11}$	55	3	7920^{12}
$GL(5,3)$	20	3	475,566,474,240
$\phi F(2,9)$	19	5	579,833,984,375,000

Table 4.4: Simple groups and related constructions

In the next chapter, we use these groups to investigate several aspects of the implementation of the random Schreier-Sims algorithm. Most tests were repeated 10 times on a Sparc Station 10/51. We give timings in seconds, averaged over those runs which found a complete BSGS.

Chapter 5

Investigating performance

In this chapter, we consider four topics related to the implementation of the random Schreier-Sims algorithm. Our discussion of methods for generating random elements and stopping conditions pertains to both permutation and matrix group versions of the algorithm. Next, we describe several sets on which matrix groups have a natural action, and discuss which are useful for the random Schreier-Sims algorithm. Finally, we describe new techniques which significantly extend the range of application of the random Schreier-Sims algorithm for matrices, by finding particular points which have small orbits.

5.1 Random elements

The generation of random group elements is an important aspect of the random Schreier-Sims algorithm, for both matrix and permutation groups. If we already know a base and strong generating set, we can produce uniform random elements by the method described in Section 2.6. This is useful if we want a smaller strong generating set or a strong generating set with respect to a different base.

If no BSGS is known, nearly uniform random elements can be generated by the algorithm described in Babai (1991). However, this algorithm has complexity $O(n^{10}(\log q)^5)$ for matrix groups, and no practical implementation of it has yet been developed. An alternative method presented in Holt & Rees (1992) is more practical, but is not guaranteed to produce nearly uniform random elements. We now present a similar method which is used in our implementation of the random

Schreier-Sims algorithm. It is based on a GAP program written by Leedham-Green and O'Brien, as well as a number of suggestions made by Cannon in personal communication.

Our algorithm for generating random group elements is based on the evaluation of random words in the strong generators. If the random words we choose are relatively short, we may only produce elements from a small part of the group. However, both matrix and permutation multiplications involve a large number of operations, so evaluating long words is time consuming. This problem is solved by storing some of the evaluated words which we have previously obtained. When we need a new random element, we compute the product of one of these stored elements with either another stored element or a strong generator. We multiply by strong generators occasionally to ensure that they all occur in the random words produced. While this method only requires a single multiplication for each evaluated word, the length of the word grows exponentially.

The algorithm presented below is in two parts: the *initialise* procedure which is called once before generating random elements; and the *random_element* function which then returns a random element. It is controlled by three predetermined parameters:

- *num* — the number of evaluated words to store;
- *len* — the length of the words after executing *initialise*;
- *prob* — the probability of multiplying by another evaluated word rather than a strong generator.

The procedure *initialise* may reset *num*, so that it is at least as large as the initial number of strong generators; this allows us to ensure that every strong generator appears in at least one of the initial words. The other two parameters remain constant. Evaluated words are stored in the array

$$w : \{0, 1, \ldots, num - 1\} \to G.$$

The variable *current* is the position in w of the evaluated word we are currently considering. Note that in *random_element* we randomly decide whether to pre- or post-multiply $w(current)$ by the group element x.

Algorithm 5.1.1 *Random element generation*

```
procedure initialise(S, var w, var num, len)
(* input:   partial strong generating set S = {s_0, s_1, ..., s_{m-1}},
            integers num and len.
   output:  array w of num evaluated words of length len. *)
begin
    let num = max(num, m);
    for current = 0 to num - 1 do
        let j_1 = current mod m;
        (* each element of S must occur in at least one word *)
        let j_2, j_3, ..., j_len be random integers in {0, 1, ..., m - 1};
        let w(current) = s_{j_1} · s_{j_2} · ··· · s_{j_len};
    end for;
end procedure.

function random_element(S, var w, num, prob)
(* input:   partial strong generating set S = {s_0, s_1, ..., s_{m-1}},
            array w of num evaluated words, probability prob.
   output:  return random element. *)
begin
    let current = (current + 1) mod num;
    let r be a random real number in [0, 1];
    if r ≤ prob then
        let i be a random integer in {0, 1, ..., num - 1};
        let x = w(i);
    else
        let i be a random integer in {0, 1, ..., m - 1};
        let x = s_i;
    end if;
    let w(current) = w(current) · x or x · w(current);
    return w(current);
end function.
```

In Table 5.1, we present results demonstrating the impact of varying the three parameters on the performance of our implementation of the random Schreier-Sims algorithm. The second stopping condition described in Section 5.2 was used with $C = 10$. The column labelled "/10" contains the number of runs out of 10 which gave a complete BSGS. Tests were carried out on a soluble group with 14 generators and a "related construction" with 3 generators. These results indicate that the random Schreier-Sims algorithm is more reliable if we increase the initial lengths of the words; this makes the random elements more uniform. However, the time taken also increases.

Group	n	q	num	len	$prob$	Time	/10
$GL(6,2)$–10–2^4	96	2	14	2	0.75	2720	5
			14	10	0.75	2839	9
			14	20	0.75	3099	10
			28	2	0.75	2598	4
			14	2	0.25	3370	2
			14	2	0.99	2665	5
$(3^2{:}4\times A_6)\cdot 2$	45	7	5	2	0.75	305	4
			5	10	0.75	319	10
			5	20	0.75	323	10
			10	2	0.75	312	7
			5	2	0.25	309	9
			5	2	0.99	312	6

Table 5.1: Investigating random element generation

The first implementation of the random Schreier-Sims algorithm by Leon (1980a) stored a single evaluated word, which was initialised to be a generator and multiplied by another generator to produce each new random element. The algorithm described by Holt & Rees (1992) stored about 10 evaluated words whose initial length was around 30; they were producing random matrices for a group recognition algorithm in which uniformness was more critical than it is with the random Schreier-Sims algorithm. Following a suggestion by Cannon, we use the following values as our default:

$$num = 5, \ len = 2, \ prob = 0.75;$$

observe from Table 5.1 that these are not necessarily optimal.

5.2 Stopping conditions

The choice of stopping condition is a very important aspect of the random Schreier-Sims algorithm. Recall, from Section 3.4, that to ensure that the algorithm terminates, we stop testing new random elements when some predetermined condition becomes true. If we know the order of the group, we can choose a stopping condition which makes the algorithm deterministic. Otherwise we need a stopping condition which maximises the probability that a complete BSGS will be constructed, while minimising the number of random elements considered.

The three most common stopping conditions are:

1. R random elements have been considered;
2. C consecutive random elements have all stripped to the identity;
3. the product of the lengths of the partial basic orbits has reached L;

where R, C and L are positive integers chosen in advance. The first condition allows us to calculate an upper bound on the running time of the algorithm, provided we have a bound on the lengths of the basic orbits. The second allows us to estimate the probability that the partial BSGS produced is complete, as shown below. If the order of the group is known in advance, we set L to be the order and use the third stopping condition.

Combinations of these conditions can also be used. The implementation of the random Schreier-Sims algorithm in Cayley asks for numbers R and C and then terminates when either (1) or (2) is true. If we have a lower bound L on the group order, then we can choose an integer C and stop when both (2) and (3) are true. Similarly, if we have an upper bound L, we choose C and stop when either (2) or (3) is true.

The following result, presented in Leon (1980b), allows us to estimate the probability that our partial BSGS is complete when the second stopping condition is used.

Theorem 5.2.1 *If B is a partial base and S is a partial strong generating set for G, then* $\left|U^{(k)} \cdot U^{(k-1)} \cdot \ldots \cdot U^{(1)}\right| = \left|\Delta^{(k)}\right| . \left|\Delta^{(k-1)}\right| . \ldots . \left|\Delta^{(1)}\right|$ *divides* $|G|$.

Proof:

It is easily seen that

$$\left|U^{(k)} \cdot U^{(k-1)} \cdot \ldots \cdot U^{(1)}\right| = \left|\Delta^{(k)}\right| . \left|\Delta^{(k-1)}\right| . \ldots . \left|\Delta^{(1)}\right| .$$

Since $\left|\Delta^{(i)}\right| = \left|H^{(i)} : H^{(i)}_{\beta_i}\right|$, we have

$$\begin{aligned} |G| &= \prod_{i=1}^{k} \left|H^{(i)} : H^{(i+1)}\right| . \left|H^{(k+1)}\right| \\ &= \left|H^{(k+1)}\right| . \prod_{i=1}^{k} \left|H^{(i)} : H^{(i)}_{\beta_i}\right| . \left|H^{(i)}_{\beta_i} : H^{(i+1)}\right| \\ &= \left|H^{(k+1)}\right| . \prod_{i=1}^{k} \left|\Delta^{(i)}\right| . \left|H^{(i)}_{\beta_i} : H^{(i+1)}\right| , \end{aligned}$$

so $\left|\Delta^{(k)}\right| . \left|\Delta^{(k-1)}\right| . \ldots . \left|\Delta^{(1)}\right|$ divides $|G|$. □

If $U^{(k)} \cdot U^{(k-1)} \cdot \ldots \cdot U^{(1)}$ is not the entire group, then it contains at most half of the elements of G. Suppose $g \in G$ has residue $\bar{g}$ with respect to our partial BSGS. Now $\bar{g} = e$ if and only if

$$g \in U^{(k)} \cdot U^{(k-1)} \cdot \ldots \cdot U^{(1)};$$

so, if g is a uniform random element of G and our partial BSGS is not complete, there is a probability of at least $1/2$ that $\bar{g} \neq e$. Hence, if C consecutive random elements strip to the identity, the probability that our strong generating set is still not complete is at most 2^{-C}. Since we are not using uniform random elements, this calculation is not exact, but it can still be used as a rough guide in choosing C.

Note that the set $U^{(k)} \cdot U^{(k-1)} \cdot \ldots \cdot U^{(1)}$ considered in Theorem 5.2.1 is not necessarily a subgroup of G, so using a partial BSGS instead of a complete one can have unpredictable results.

Our implementation was tested with stopping conditions (2) and (3). Table 5.2 gives timings for condition (2) with $C = 10$, 30 or 50, and for condition (3) with L equal to the order of the group. In addition, the columns labelled "/10" contain the number of executions out of ten which yielded a complete

Group	n	q	$C = 10$		$C = 30$		$C = 50$		$L = \|G\|$
			time	/10	time	/10	time	/10	time
$GL(2,7)$–3–3^3	54	7	1449	10	1636	10	1767	10	1497
$GL(4,3)$–1–3^3	108	3	1970	9	3074	10	4367	10	2007
$GL(4,3)$–50–3^3	108	3	24211	10	26167	10	29166	10	22284
$GL(5,3)$–2–3^3	135	3	12102	8	16629	10	22310	10	8093
$GL(5,3)$–9–2^4	80	3	6550	7	7214	10	7942	10	6469
S_{30}	29	7	188	3	224	5	270	9	211
S_{30}	29	31	208	2	199	6	239	8	169
S_{50}	49	53	1948	1	2824	6	3297	7	3036
S_{57}	55	3	5052	2	6131	7	5853	9	6562
S_{63}	62	53	8441	1	10418	2	11213	6	10198
S_{65}	64	11	—	0	10984	1	13917	4	10061
A_8	20	11	40	10	40	10	42	10	38
A_8	28	11	68	10	74	10	76	10	68
A_8	64	11	319	10	360	10	412	10	313
M_{11}	24	3	10	5	14	9	19	10	10
M_{11}	44	2	44	10	62	10	84	10	42
M_{11}	44	7	47	10	64	10	88	10	45
M_{11}	55	7	55	5	103	10	154	10	52
M_{12}	55	7	809	10	850	10	902	10	810
M_{12}	120	17	15475	10	16127	10	16712	10	15588
M_{22}	21	7	666	10	682	10	675	10	672
$2.M_{22}$	34	2	—	0	193	10	222	10	180
J_1	14	11	140	10	141	10	141	10	141
J_1	27	11	276	6	283	10	290	10	281
$2.J_2$	36	3	6183	10	6252	10	6244	10	6268
$2.J_2$	42	3	—	0	5108	9	5363	10	5143
$4.J_2$	12	3	350	1	396	10	394	10	394
$6.J_3$	18	2	—	0	453	4	477	8	437
$U_4(2)$	58	5	247	10	276	10	348	10	243
$U_4(2)$	64	2	80	5	129	10	197	10	49
$U_4(2)$	81	11	172	4	339	10	478	10	145
$(3^2{:}4\times A_6)\cdot 2$	18	7	61	6	63	10	64	10	61
$(3^2{:}4\times A_6)\cdot 2$	27	7	122	6	126	10	131	10	122
$(3^2{:}4\times A_6)\cdot 2$	45	7	305	4	324	10	343	10	307
$Sz(8)$	65	29	55	4	111	10	161	10	37
$4.Suz$	12	3	—	0	160	6	160	10	154
$A_5 \times A_5$	25	7	16	10	18	10	20	10	15
$M_{11} \wr M_{11}$	55	3	3297	10	3436	10	3790	9	3253
$GL(5,3)$	20	3	11	10	15	10	18	10	9
$\phi F(2,9)$	19	5	14	10	18	10	22	10	12

Table 5.2: Comparison of stopping conditions

base and strong generating set. These results demonstrate that $C = 10$ is far too small for us to have any confidence in the partial BSGS produced, $C = 30$ is sufficient for most groups, and $C = 50$ gives us a complete BSGS in almost every case. When the third stopping condition is used, the random Schreier-Sims algorithm almost always outperforms the Schreier-Sims and Todd-Coxeter Schreier-Sims algorithms.

Clearly, considering a larger number of random elements increases the probability that our BSGS is complete. However, a comparison with Table 5.1 suggests that a more efficient method of achieving this goal is to generate random elements which are more uniform. All the timings reported in the next two sections are for stopping condition (2) with $C = 50$.

5.3 Point sets

The most natural action of a matrix group is on the underlying vector space, as we saw in Section 4.1. In this section, we consider several other point sets for matrix groups and discuss their usefulness for the random Schreier-Sims algorithm. The properties that make a point set useful for this algorithm include:

- There must be an efficient method for calculating the image of a point under the action of a matrix.
- It must be easy to determine if two points are identical; ideally, we want a canonical representation for each point.
- The memory requirements of a point should be comparable to those of a vector.
- The action of the group on the point set should produce small orbits which can be found easily.

Suppose $G \leq GL(n, q)$. One of the most obvious types of point is a set $W \subseteq V(n, q)$ with the action of $A \in G$ defined by

$$W^A = \{wA : w \in W\}.$$

Such sets occupy more memory than single vectors and their images take longer to calculate. While they may have shorter orbits, this cannot compensate for the increased memory required for each point. If $w \in W$, then clearly every vector in w^G is contained in at least one of the sets in W^G; hence, to store the orbit of W, we must store at least as many vectors as are contained in the union of the orbits of the elements of W. A similar argument applies for ordered tuples of vectors.

Subspaces of $V(n, q)$ are more useful than arbitrary sets. We need only store a basis, and we can calculate the action of a matrix by considering its action on each basis vector. The basis can be stored as the rows of a matrix, which is then row-reduced to produce a canonical representation. An efficient row-reduction algorithm is necessary if we use multi-dimensional subspaces.

Row-reduction of the matrix of a one-dimensional subspace is simply multiplication by a scalar. Hence, one-dimensional subspaces have the same memory requirements as vectors, and the action of a matrix takes only slightly longer to calculate. Suppose we have $v \in V(n, q)$ whose orbit v^G has length M. Now suppose that, instead of taking our first base point to be v, we take $\beta_1 = \langle v \rangle$ and $\beta_2 = v$. Then every element of $\Delta^{(2)} = \beta_2^{G_{\beta_1}}$ is of the form λv for some non-zero $\lambda \in GF(q)$ and it is easily seen that

$$\left\{ \lambda : \lambda v \in \Delta^{(2)} \right\}$$

is a subgroup of the multiplicative group of $GF(q)$. Hence, the length of $\Delta^{(2)}$ is a divisor d of $q - 1$ and, instead of having to calculate one orbit of length M, we calculate two orbits whose lengths are M/d and d. More generally, we can precede every vector in our base by the one-dimensional subspace containing it. This method was first suggested by Butler (1976). It is least effective for smaller fields, and of no benefit over $GF(2)$.

An arbitrary subspace of dimension higher than one is unlikely to be more useful than a one-dimensional subspace. However, eigenspaces can be useful, as we see in the next section. Sets of subspaces offer no improvement over individual subspaces, by the same argument used for sets of vectors. Other structures might be useful for particular types of groups, but we restrict our attention to point sets which are potentially useful for arbitrary matrix groups.

We now present some timings for the random Schreier-Sims algorithm using Butler's method of alternating one-dimensional subspaces and vectors. We also list the first ten basic indices. Since the basic indices sometimes vary for different runs of the algorithm, we present only one example for each group.

The results in Table 5.3 demonstrate that the random Schreier-Sims algorithm is very effective for soluble groups. Although the implementation took a long time with some of the groups, this is due to the large matrix dimension and group order, rather than the size of the orbits.

Group	n	q	Time	Basic indices				
$GL(2,7)$–15–2^4	32	7	149	64 6	8 6	16 6	8 6	4 6 ...
$GL(2,7)$–3–3^3	54	7	1767	45 2	2 1	6 2	6 2	18 2 ...
$GL(4,3)$–1–3^3	108	3	4367	135 1	15 1	15 1	45 1	15 1 ...
$GL(4,3)$–1–2^4	64	3	540	80 1	10 1	20 1	10 1	40 1 ...
$GL(4,3)$–50–3^3	108	3	29166	216 2	24 2	24 2	3 2	72 2 ...
$GL(5,3)$–2–3^3	135	3	22310	297 2	33 2	33 2	99 2	33 2 ...
$GL(5,3)$–9–2^4	80	3	7942	80 2	4 1	2 1	2 1	10 2 ...
$GL(6,2)$–10–2^4	96	2	4760	144 1	18 1	36 1	18 1	6 1 ...
$GL(6,2)$–21–3^3	162	2	68523	243 1	6 1	27 1	81 1	81 1 ...
$GL(6,2)$–33–2^3	48	2	459	288 1	2 1	36 1	6 1	2 1 ...

Table 5.3: Soluble groups

The results for almost simple groups, in Table 5.4, demonstrate that these methods are usually just as effective as for the soluble groups. However, in three cases, the algorithm failed to complete in a "reasonable" time; this is indicated by the ∞ symbol.

Group	n	q	Time	Basic indices				
S_{20}	18	5	27	190 2	18 1	17 1	16 1	15 1 ...
S_{20}	19	23	∞					
S_{25}	24	17	96	300 2	23 1	22 1	21 1	20 1 ...
S_{30}	29	7	270	435 2	28 1	27 1	26 1	25 1 ...
S_{30}	29	31	239	435 2	28 1	27 1	26 1	25 1 ...
S_{35}	34	2	∞					
S_{40}	38	2	∞					
S_{50}	49	53	3297	1225 2	48 1	47 1	46 1	45 1 ...
S_{57}	55	3	5853	1596 2	55 1	54 1	53 1	52 1 ...
S_{63}	62	53	11212	1953 2	61 1	60 1	59 1	58 1 ...
S_{65}	64	11	13917	2080 2	63 1	62 1	61 1	60 1 ...
S_{72}	70	2	16799	2556 1	140 1	69 1	68 1	67 1 ...

Table 5.4: Almost simple groups

The results in Table 5.5 show that, for most of our simple groups and related constructions, the first basic orbit is very large compared to the other orbits.

Group	n	q	Time	Basic indices					
A_8	20	11	42	20160 1					
A_8	28	11	76	20160 1					
A_8	64	11	412	20160 1					
M_{11}	24	3	19	3960 2					
M_{11}	44	2	84	7920 1					
M_{11}	44	7	88	7920 1					
M_{11}	55	7	154	3960 1	2 1				
M_{12}	120	17	16712	95040 1					
M_{12}	55	7	902	95040 1					
M_{22}	21	7	675	443520 1					
$2.M_{22}$	34	2	222	18480 1	24 1	2 1			
J_1	7	11	8	7315 2	6 1	2 1			
J_1	14	11	141	175560 1					
J_1	27	11	290	87780 2					
$2.J_2$	36	3	6244	1209600 1					
$2.J_2$	42	3	5363	60480 1	2 1				
$4.J_2$	12	3	394	201600 2	6 1				
$6.J_3$	18	2	477	130815 1	1152 1	2 1			
$U_4(2)$	58	5	348	25920 1					
$U_4(2)$	64	2	197	270 3	16 2				
$U_4(2)$	81	11	478	810 2	16 1				
$(3^2{:}4\times A_6)\cdot 2$	18	7	64	25920 3					
$(3^2{:}4\times A_6)\cdot 2$	27	7	131	25920 3					
$(3^2{:}4\times A_6)\cdot 2$	45	7	343	25920 3					
$Sz(8)$	65	29	161	65 7	64 1				
$4.Suz$	12	3	160	32760 2	1980 2	96 1	36 1	2 1	
$A_5 \times A_5$	25	7	20	3600 1					
$M_{11} \wr M_{11}$	55	3	3790	1210 2	1100 2	990 2	880 2	110 2	...
$GL(5,3)$	20	3	18	121 2	120 2	117 2	108 2	81 2	
$\phi F(2,9)$	19	5	22	190 4	5 1	5 1	5 1	5 1	...

Table 5.5: Simple groups and related constructions

5.4 Eigenvectors and eigenspaces

In the previous section, we considered different point sets on which a matrix group can act. We now consider the possibility of choosing particular points in these sets which we expect will have small orbits for some theoretical reason. The points we choose are eigenvectors and eigenspaces of the generating matrices. This differs significantly from the technique used by Butler to find small orbits, which we discussed in Section 4.3.

Definition 5.4.1 *Let A be an $n \times n$ matrix over $GF(q)$. The* characteristic polynomial *of A is*

$$c_A(x) = det(xI - A),$$

where I is the identity matrix. For each irreducible factor $g(x)$ of $c_A(x)$, there is a corresponding eigenspace

$$\{v \in V(n, q) : v.\, g(A) = 0\}.$$

An eigenvector *is a non-zero element of an eigenspace.*

Note that this definition differs slightly from the standard one, which only considers linear factors of $c_A(x)$. We consider the impact of the following selections on the performance of the random Schreier-Sims algorithm:

- Let the first base point be an eigenvector of a generator.
- Let the first base point be an eigenvector of more than one generator.
- Let the first base point be an eigenspace of a generator.
- Let several initial base points be eigenvectors of the generators.

Eigenspaces and eigenvectors can be found efficiently for matrices of large dimension; in practice, we computed them using GAP.

If $g(x)$ is linear, say $g(x) = x - \lambda$, then for any eigenvector v we have $v^A = \lambda v$, and so the orbit $v^{\langle A \rangle}$ contains at most $q - 1$ points. Hence, we expect that the orbit of v under the action of G will be shorter than the orbit of a vector chosen by the method of Section 4.3. Suppose now, for example, that $g(x) = x^2 + x + 1$, then we have $v^{A^2+A} = -v$. Since this formula involves addition of matrices as well as multiplication, we have less reason to believe that the eigenvectors will have small orbits. We take as base points eigenvectors corresponding to factors whose degree is minimal. If there is more than one such factor, we choose an eigenvector from an eigenspace of the smallest dimension.

We found a linear factor of a characteristic polynomial for every one of our simple groups and related constructions. Using a corresponding eigenvector as the first base point frequently leads to significant improvements in the performance of the implementation, as shown in Table 5.6.

Group	n	q	Deg	Dim	Time	$\|\Delta^{(1)}\|$
A_8	20	11	1	2	9	2880
A_8	28	11	1	4	19	2880
A_8	64	11	1	10	153	2880
M_{11}	24	3	1	6	10	110
M_{11}	44	2	1	12	45	1980
M_{11}	44	7	1	12	57	1980
M_{11}	55	7	1	13	100	1980
M_{12}	55	7	1	18	360	31680
M_{12}	120	17	1	40	2868	31680
M_{22}	21	7	1	6	228	110880
$2.M_{22}$	34	2	1	20	194	18480
J_1	7	11	1	1	5	5852
J_1	14	11	1	2	17	12540
J_1	27	11	1	3	59	12540
$2.J_2$	36	3	1	5	389	37800
$2.J_2$	42	3	1	7	1557	151200
$4.J_2$	12	3	1	4	35	12600
$6.J_3$	18	2	1	6	167	61560
$U_4(2)$	58	5	1	14	101	6480
$U_4(2)$	64	2	1	16	223	6480
$U_4(2)$	81	11	1	21	399	6480
$(3^2{:}4\times A_6)\cdot 2$	18	7	1	2	8	1080
$(3^2{:}4\times A_6)\cdot 2$	27	7	1	3	23	2160
$(3^2{:}4\times A_6)\cdot 2$	45	7	1	2	71	1080
$Sz(8)$	65	29	1	16	331	7280
$4.Suz$	12	3	1	6	141	32760
$A_5 \times A_5$	25	7	1	5	8	720
$M_{11} \wr M_{11}$	55	3	1	1	3265	121
$GL(5,3)$	20	3	1	6	18	121
$\phi F(2,9)$	19	5	1	1	23	1

Table 5.6: Using eigenvectors of a generator

Using eigenvectors of a single generator guarantees a small orbit under the action of that matrix, but not the others. Sometimes, a vector is an eigenvector of more than one generator. The results of using eigenvectors of two generators are shown in Table 5.7; these were found by calculating the intersections of the eigenspaces in GAP. A single eigenvector corresponding to a linear factor of the characteristic polynomial is generally more effective than a vector in two eigenspaces of non-linear factors. Clearly, if the matrices are of large dimension, then the eigenspaces are likely to have large dimension, so we are more likely to find a non-trivial intersection.

Group	n	q	Deg 1	Dim 1	Deg 2	Dim 2	Time	$\|\Delta^{(1)}\|$
A_8	20	11	3	9	3	9	11	1680
A_8	28	11	3	12	3	12	77	20160
A_8	64	11	3	27	3	27	412	20160
M_{11}	44	7	1	20	2	20	53	660
M_{11}	55	7	1	27	2	28	131	660
M_{12}	120	17	1	56	2	80	2750	47520
M_{22}	21	7	1	13	2	8	21	2310
$6.J_3$	18	2	1	9	2	12	376	130815
$U_4(2)$	81	11	1	39	2	36	475	2160
$(3^2{:}4\times A_6)\cdot 2$	27	7	2	6	2	14	14	180
$(3^2{:}4\times A_6)\cdot 2$	45	7	2	8	2	24	62	180
$4.Suz$	12	3	1	6	2	12	149	32760
$A_5 \times A_5$	25	7	1	5	1	5	8	36
$GL(5,3)$	20	3	5	5	5	5	361	28314

Table 5.7: Using eigenvectors of two generators

We also investigated setting the first base point to be a complete eigenspace. If the eigenspace is of dimension one, this is the same as using an eigenvector, since all vectors are preceded by the corresponding one-dimensional subspace. If the dimension of the eigenspace is large, the cost of the row-reductions makes this method impractical. In Table 5.8, we give the results of using eigenspaces of dimension between two and five. Note that we chose the eigenspace of the eigenvector used in Table 5.6. Often the reduction in orbit size does not compensate for the increased time taken to calculate images, but this reduction may sometimes allow us to calculate a BSGS for groups where we could not otherwise do so.

Group	n	q	Deg	Dim	Time	$\lvert\Delta^{(1)}\rvert$
A_8	20	11	1	2	11	960
A_8	28	11	1	4	37	960
J_1	14	11	1	2	18	4180
J_1	27	11	1	3	88	4180
$2.J_2$	36	3	1	5	705	9450
$4.J_2$	12	3	1	4	8	315
$(3^2{:}4\times A_6)\cdot 2$	27	7	1	3	38	1080
$(3^2{:}4\times A_6)\cdot 2$	18	7	1	2	13	1080
$(3^2{:}4\times A_6)\cdot 2$	45	7	1	2	94	1080
$A_5 \times A_5$	25	7	1	5	9	72
$GL(5,3)$	20	3	1	6	27	121

Table 5.8: Using eigenspaces

Group	n	q	Normal	Eigenvector	Intersection	Eigenspace
A_8	20	11	20160	2880	1680	960
A_8	28	11	20160	2880	20160	960
A_8	64	11	20160	2880	20160	—
M_{11}	24	3	3960	110	—	—
M_{11}	44	2	7920	1980	—	—
M_{11}	44	7	7920	1980	660	—
M_{11}	55	7	3960	1980	660	—
M_{12}	55	7	95040	31680	—	—
M_{12}	120	17	95040	31680	47520	—
M_{22}	21	7	443520	110880	2310	—
$2.M_{22}$	34	2	18480	18480	—	—
J_1	7	11	7315	5852	—	—
J_1	14	11	175560	12540	—	4180
J_1	27	11	87780	12540	—	4180
$2.J_2$	36	3	1209600	37800	—	9450
$2.J_2$	42	3	60480	15120	—	—
$4.J_2$	12	3	201600	12600	—	315
$6.J_3$	18	2	130815	61560	130815	—
$U_4(2)$	58	5	25920	6480	—	—
$U_4(2)$	64	2	270	6480	—	—
$U_4(2)$	81	11	810	6480	2160	—
$(3^2{:}4\times A_6)\cdot 2$	18	7	25920	1080	—	1080
$(3^2{:}4\times A_6)\cdot 2$	27	7	25920	2160	180	1080
$(3^2{:}4\times A_6)\cdot 2$	45	7	25920	1080	180	1080
$Sz(8)$	65	29	65	7280	—	—
$4.Suz$	12	3	32760	32760	32760	—
$A_5 \times A_5$	25	7	3600	720	36	72
$M_{11} \wr M_{11}$	55	3	1210	121	—	—
$GL(5,3)$	20	3	121	121	28314	121
$\phi F(2,9)$	19	5	190	1	—	—

Table 5.9: First basic indices for simple groups

Table 5.9 is a summary of the sizes of the first basic orbits when these different techniques are used. Note that "Normal" indicates that the first point was chosen by the algorithm in Section 4.3, and "Intersection" indicates that the point is in the intersection of eigenspaces of two different generators.

With simple groups, we found that significant improvements in effectiveness could be made by letting the first base point be an eigenvector. However, for our almost simple groups, this just makes the first orbit very small without affecting subsequent orbits, which are sometimes very large. Instead, we took a number of the initial base points to be eigenvectors. Two cases occurred with our almost simple groups: when q does not divide n, every generator has a one-dimensional eigenspace and an eigenspace of dimension $n-1$; otherwise, every generator has a single eigenspace of dimension $n-1$. In the first case we took our base points to be elements of these one-dimensional eigenspaces. Alternatively, we calculated the intersection of $n-2$ of the eigenspaces (there are $n-1$ generators). This intersection has dimension one; we took our first base point from it. Our second eigenvector was chosen from an intersection of $n-3$ eigenspaces and so on. The results of this technique are shown in Table 5.10.

Group	n	q	Time	Basic indices				
S_{20}	19	23	48	190 2	18 1	17 1	16 1	15 1 ...
S_{35}	34	2	59	35 1	34 1	33 1	32 1	31 1 ...
S_{40}	38	2	290	780 1	38 1	37 1	36 1	35 1 ...
S_{57}	55	3	4943	1596 2	55 1	54 1	53 1	52 1 ...
S_{63}	62	53	9677	1953 2	61 1	60 1	59 1	58 1 ...
S_{65}	64	11	11826	2080 2	63 1	62 1	61 1	60 1 ...

Table 5.10: Almost simple groups

In summary, the techniques described in this section significantly improve the performance of the random Schreier-Sims algorithm. These techniques will be even more effective for the basic Schreier-Sims algorithm, since they will reduce the number of Schreier generators considered. We expect that they should be useful for the other variations of the algorithm. We have also extended the range of application of the algorithm for computing with certain almost simple groups, and anticipate that similar techniques will be useful for other groups in this category.

Bibliography

M. Aschbacher (1984), "On the maximal subgroups of the finite classical groups", *Invent. Math.*, **76**, 469–514.

László Babai (1991), "Local expansion of vertex-transitive graphs and random generation in finite groups", *Theory of Computing*, (Los Angeles, 1991), pp. 164–174. Association for Computing Machinery, New York.

Wieb Bosma and John Cannon (1992), "Structural computation in finite permutation groups", *CWI Quarterly*, **5**(2), 127–160.

Gregory Butler (1976), "The Schreier Algorithm for Matrix Groups", SYMSAC '76, *Proc. ACM Sympos. symbolic and algebraic computation*, (New York, 1976), pp. 167–170. Association for Computing Machinery, New York.

Gregory Butler (1979), *Computational Approaches to Certain Problems in the Theory of Finite Groups*, PhD thesis. University of Sydney.

Gregory Butler (1982), "Computing in Permutation and Matrix Groups II: Backtrack Algorithm", *Math. Comp.*, **39**, 671–680.

Gregory Butler (1986), "Data Structures and Algorithms for Cyclically Extended Schreier Vectors", *Congr. Numer.*, **52**, 63–78.

G. Butler (1991), *Fundamental Algorithms for Permutation Groups*, Lecture Notes in Comput. Sci., **559**. Springer-Verlag, Berlin, Heidelberg, New York.

Gregory Butler and John J. Cannon (1982), "Computing in Permutation and Matrix Groups I: Normal Closure, Commutator Subgroups, Series", *Math. Comp.*, **39**, 663–670.

John Cannon and George Havas (1992), "Algorithms for Groups", *Australian Computer Journal*, **24**(2), 51–60.

John J. Cannon (1984), "An Introduction to the Group Theory Language, Cayley", *Computational Group Theory*, (Durham, 1982), pp. 145–183. Academic Press, London, New York.

J.H. Conway, R.T. Curtis, S.P. Norton, R.A. Parker and R.A. Wilson (1985), *Atlas of finite groups*. Clarendon Press, Oxford.

Marshall Hall, Jr. (1959), *The Theory of Groups*. Macmillan Co., New York.

George Havas (1991), "Coset enumeration strategies", *International Symposium on Symbolic and Algebraic Computation* '91, pp. 191–199. ACM Press, New York.

George Havas, J.S. Richardson and Leon S. Sterling (1979), "The last of the Fibonacci groups", *Proc. Roy. Soc. Edinburgh Sect. A*, **83A**, 199–203.

Derek F. Holt and Sarah Rees (1992), "An implementation of the Neumann–Praeger algorithm for the recognition of special linear groups", *J. Experimental Math.*, **1**, 237–242.

Derek F. Holt and Sarah Rees (1994), "Testing modules for irreducibility", *J. Austral. Math. Soc. Ser. A*, **57**.

D.L. Johnson (1990), *Presentations of Groups*, London Math. Soc. Stud. Texts, **15**. Cambridge University Press, Cambridge.

Brian W. Kernighan and Dennis M. Ritchie (1988), *The C programming language* (2nd edition). Prentice-Hall Internat. Ser. Comput. Sci., Englewood Cliffs, NJ.

Donald E. Knuth (1973), *The art of computer programming. Volume 3: Sorting and Searching*. Addison-Wesley, Massachusetts.

Jeffrey S. Leon (1980a), "Finding the order of a permutation group", Bruce Cooperstein and Geoffrey Mason (Eds.), *Finite Groups*, Proc. Sympos. Pure Math., **37**, (Santa Cruz, 1979), pp. 511–517. Amer. Math. Soc., Providence, RI.

Jeffrey S. Leon (1980b), "On an algorithm for finding a base and strong generating set for a group given by generating permutations", *Math. Comp.*, **20**, 941–974.

Jeffrey S. Leon (1991), "Permutation group algorithms based on partitions, I: Theory and algorithms", *J. Symbolic Comput.*, **12**, 533–583.

Stephen Alexander Linton (1989), *The maximal subgroups of the sporadic groups Th, Fi_{24} and Fi'_{24} and other topics*, PhD thesis. University of Cambridge.

J. Neubüser (1982), "An elementary introduction to coset table methods in computational group theory", C.M. Campbell and E.F. Robertson (Eds.), *Groups – St Andrews 1981*, London Math. Soc. Lecture Note Ser., **71**, pp. 1–45. Cambridge University Press, Cambridge.

Peter M. Neumann and Cheryl E. Praeger (1992), "A recognition algorithm for special linear groups", *Proc. London Math. Soc. (3)*, **65**, 555–603.

Martin Schönert *et al.* (1993), GAP – *Groups, Algorithms and Programming.* Lehrstuhl D. für Mathematik, RWTH, Aachen.

M.W. Short (1992), *The Primitive Soluble Permutation Groups of Degree less than* 256, Lecture Notes in Math., **1519**. Springer-Verlag, Berlin, Heidelberg, New York.

Charles C. Sims (1970), "Computational methods in the study of permutation groups", *Computational problems in abstract algebra*, (Oxford, 1967), pp. 169–183. Pergamon Press, Oxford.

Charles C. Sims (1971), "Determining the conjugacy classes of permutation groups", Garret Birkhoff and Marshall Hall, Jr. (Eds.), *Computers in algebra and number theory*, Proc. Amer. Math. Soc., **4**, (New York, 1970), pp. 191–195. Amer. Math. Soc., Providence, RI.

Charles C. Sims (1978), "Some group theoretic algorithms", A. Dold and B. Eckmann (Eds.), *Topics in algebra*, Lecture Notes in Math., **697**, (Canberra, 1978), pp. 108–124. Springer-Verlag, Berlin, Heidelberg, New York.

Michio Suzuki (1982), *Group Theory I*, Grundlehren Math. Wiss., **247**. Springer-Verlag, Berlin, Heidelberg, New York.

J.A. Todd and H.S.M. Coxeter (1936), "A practical method for enumerating cosets of a finite abstract group", *Proc. Edinburgh Math. Soc.*, **5**, 26–34.

Helmut Wielandt (1964), *Finite permutation groups.* Academic Press, New York.

Index