MATHEMATICS OF COMPUTATION, VOLUME 35, NUMBER 151
JULY 1980, PAGES 941–974

On an Algorithm for Finding a Base and a Strong Generating Set for a Group Given by Generating Permutations

By Jeffrey S. Leon[1]

Abstract. This paper deals with the problem of finding a base and strong generating set for the group generated by a given set of permutations. The concepts of base and strong generating set were introduced by Sims [5], [6] and provide the most effective tool for computing with permutation groups of high degree. One algorithm, originally proposed by Sims [7], is described in detail; its behavior on a number of groups is studied, and the influence of certain parameters on its performance is investigated. Another algorithm, developed by the author, is given, and it is shown how the two algorithms may be combined to yield an exceptionally fast and effective method.

1. Introduction. An important problem in computational group theory involves the "determination" of the group G generated by given permutations $s_1, s_2, \ldots, s_m$ on a finite set Ω. As G may be quite large even when Ω is relatively small, it is seldom feasible to produce an explicit list of the elements of G. A more realistic goal is to determine properties of the group G. We would like, at least, to be able to answer such questions as:

(i) What is the order of G?

(ii) Given a permutation g on Ω, is $g \in G$? If so, write g as a word in $s_1, s_2, \ldots, s_m$.

(iii) How can an element be selected at random from G?

If $\{s_1, s_2, \ldots, s_m\}$ is an arbitrary generating set for G, these questions may be difficult to answer even when $|\Omega|$ is quite small. To overcome this difficulty, Sims [5], [6] introduced the concepts of base and strong generating set. Let $G_{\alpha_1\alpha_2\cdots\alpha_k}$ denote the (pointwise) stabilizer in G of $\alpha_1, \alpha_2, \ldots, \alpha_k$. A subset $\{\beta_1, \beta_2, \ldots, \beta_k\}$ of Ω is a *base* for G on Ω, if $|G_{\beta_1\beta_2\cdots\beta_k}| = 1$. It follows that two elements of G which agree on a base are identical. A subset T of the group G is a *strong generating set* for G relative to the ordered base $(\beta_1, \beta_2, \ldots, \beta_k)$, if $G_{\beta_1\beta_2\cdots\beta_{i-1}}$ is generated by $G_{\beta_1\beta_2\cdots\beta_{i-1}} \cap T$ for $i = 1, 2, \ldots, k$. Thus a base and strong generating set provide generators for each group in the chain

$$1 = G_{\beta_1\beta_2\cdots\beta_k} \subseteq G_{\beta_1\beta_2\cdots\beta_{k-1}} \subseteq \cdots \subseteq G_{\beta_1} \subseteq G.$$

Received January 26, 1979.

1980 *Mathematics Subject Classification.* Primary 20–04, 20B99, 20F05.

[1] Work partially supported by NSF grant MCS76-03143. Computing services used in this research were provided by the Computer Center of the University of Illinois at Chicago Circle. Their assistance is gratefully acknowledged.

0025-5718/80/0000-0120/$09.50

In some circumstances, this makes the problem of computing in G dependent on $\Sigma_{i=0}^{k-1} |G_{\beta_1\beta_2\cdots\beta_i} : G_{\beta_1\beta_2\cdots\beta_{i+1}}|$ rather than $|G|$.

The concepts of base and strong generating set are most useful when the base size is small compared to the degree. Not every group has such a base; for example, the symmetric group S_n on n letters has no base with fewer than $n - 1$ points. However, a great many interesting groups, especially primitive groups, have very small bases—often fewer than ten points. For example, the Janko sporadic simple groups J_1, J_2, and J_3 in their representations of degrees 276, 100, and 6156 have bases with 3, 4, and 3 points, respectively. Given a small base for G on Ω and a strong generating set relative to this base, questions (i), (ii), and (iii) are very easy to answer computationally even when $|\Omega|$ is quite large (thousands or tens of thousands); Section 5 gives details. In fact, Sims has developed algorithms for computing centralizers of elements [5], conjugacy classes [5], and intersections [6] in groups with a known base and strong generating set.

Unfortunately, the original set $\{s_1, s_2, \ldots, s_m\}$ of permutations seldom forms a strong generating set (relative to some base) for the group G that it generates. Usually it is necessary to add additional permutations to the set in order to obtain strong generation. On occasion it is desirable to specify an initial segment $\beta_1, \beta_2, \ldots, \beta_k$ of the base. This leads to the fundamental problem:

(*) Given a set $\{s_1, s_2, \ldots, s_m\}$ of permutations on Ω and an ordered subset $(\beta_1, \beta_2, \ldots, \beta_k)$ of Ω, extend $(\beta_1, \beta_2, \ldots, \beta_k)$ to a base for $G = \langle s_1, s_2, \ldots, s_m \rangle$ and extend $\{s_1, s_2, \ldots, s_m\}$ to a strong generating set relative to this base.

Sims' original approach (see [6]) to this problem was based on a theorem of Schreier [3, Lemma 7.2.2] giving a generating set for a subgroup of a group. Unfortunately, the Schreier generating set tends to be quite large, and this method, which I shall call the Schreier-Sims method, requires an amount of time proportional at least to the square of the degree (assuming the group is transitive); it seems to be limited, except under special circumstances, to degrees of a few hundred.

Recently Sims suggested another approach (see [7] and [8]) combining aspects of the original Schreier-Sims method with an interruptible Todd-Coxeter coset enumeration algorithm. This method, which I shall call the Schreier-Todd-Coxeter-Sims (or STCS) method, is considerably more complicated than the Schreier-Sims method; so far, at least, no good theoretical bound on its time requirements has been obtained, due in part to the unpredictability of coset enumeration. However, the new method seemed likely to outperform the Schreier-Sims method; moreover, it yielded not only a base and strong generating set but also a set of defining relators for the group in terms of the strong generators. Sims coded a version of the new algorithm in the computer language APL; however, limitations in the APL language prevented testing it on groups with degrees much over one hundred, at which differences between it and the Schreier-Sims method might become apparent.

The author has developed a fast interruptible coset enumeration program, coded

in FORTRAN (with sections in assembler language). He has also developed a version of the Schreier-Todd-Coxeter-Sims algorithm (in FORTRAN), which uses this coset enumeration program. This version of STCS has been tested on a number of groups with degrees between 100 and 10000. It contains several input parameters which control the manner in which coset enumeration is used. The influence of these parameters on performance of the algorithm has been investigated, and an attempt has been made to estimate optimal values for them. Performance turns out to vary considerably with the values selected for the parameters. With appropriate choices, the STCS algorithm seems to be quite effective on groups of degree 10000 or higher, provided that a small base exists. It is especially fast when used to verify strong generation—that is, when the original set $(\beta_1, \beta_2, \ldots, \beta_k)$ of (*) turns out already to be a base for G on Ω, and the original set $\{s_1, s_2, \ldots, s_m\}$ turns out to be a strong generating set relative to $(\beta_1, \beta_2, \ldots, \beta_k)$. When additional base points or generators are required, the algorithm is somewhat slower (but still relatively fast). Unfortunately, memory requirements of STCS are at least double those of the Schreier-Sims method; however, it may be possible to modify the STCS method in order to reduce memory usage to approximately that of Schreier-Sims, at the cost of slightly increased execution times.

The author has developed still another method for extending $\mathcal{B} = \{\beta_1, \beta_2, \ldots, \beta_k\}$ and $S = \{s_1, s_2, \ldots, s_m\}$. This approach, called the random Schreier method, combines aspects of Schreier-Sims with selection of (hopefully) random elements from the group G. Unlike the other two methods, it yields merely a probable base and a probable strong generating set; however, it requires exceedingly little time, and the STCS algorithm may be used to verify strong generation (where STCS performs best) and, if necessary, to further extend $\mathcal{B}$ and S. The random Schreier method may be programmed to select involutory generators in many cases.[2] When it is finished, redundant strong generators may be removed using a technique described by Sims (see [6, p. 24]). Thus the relatively high memory requirements of STCS can be reduced by minimizing $|S|$. Memory requirements of the random Schreier method are comparable to those of Schreier-Sims. Experimental evidence suggests that the most efficient way of extending $\mathcal{B}$ and S to a base and strong generating set is to apply the random Schreier method, to eliminate redundant strong generators, and then to apply the STCS method.

Section 2 summarizes the notation used here. Sections 3, 4, and 5 present background material necessary for the STCS and random Schreier methods. Section 3 discusses interruptible coset enumeration, Section 4 gives some simple algorithms for orbit computations in permutation groups, and Section 5 summarizes those aspects of the Sims theory of bases and strong generating sets which will be needed here; an analogous concept, called a strong set of defining relators, also is presented in Section 5. Section 6 describes the author's version of the Schreier-Todd-Coxeter-Sims algorithm, and Section 7 presents the random Schreier method. Substantial portions of Sections 3, 5, and 6

[2] Many algorithms for computing in permutation groups (including STCS) require generating sets closed under inversion. In this case, an involutory generator requires only half as much memory as a noninvolutory one.

are based on work of Sims; some of this work may be found in references [5] and [6] and in the unpublished notes [7]. Sections 8 and 9 discuss problems in implementing interruptible coset enumeration and the STCS algorithm, respectively, on a computer. Section 10 contains experimental results on the performance of the STCS algorithm; in particular, the influence of the input parameters on performance is studied.

2. Definitions and Notation. This section summarizes the notation used in this article. Let

$Z^+ = \{0, 1, 2, \ldots\}$, $Z^- = \{0, -1, -2, \ldots\}$,

$\varnothing$ = empty set,

$|X|$ = cardinality of the set X,

S_Ω = symmetric group on the set Ω.

If G is a permutation group on Ω, let

γ^g = image of point γ in Ω under permutation g of G (permutations will act on the right; $\gamma^{(gh)} = (\gamma^g)^h$),

$\gamma^G = \{\gamma^g \mid g \in G\}$ = G-orbit of γ,

$\Gamma^G = \{\gamma^g \mid g \in G, \gamma \in \Gamma\}$,

$G_{\beta_1\beta_2\cdots\beta_k}$ = pointwise stabilizer in G of points $\beta_1, \beta_2, \ldots, \beta_k$ of Ω,

$\langle T \rangle$ = subgroup of G generated by the subset T of G,

1 = identity element of G,

$|g|$ = order of the element g of G.

If X is a set, let

$\mathcal{W}(X)$ = set of all words in X,

$\ell(w)$ = length of the word $w \in \mathcal{W}(X)$,

e = trivial word ($\ell(e) = 0$).

If $x \longrightarrow x'$ is an involutory map on the set X ($(x')' = x$), let

$F(X, ')$ = group generated by X subject only to the relators xx', $x \in X$,

$[w]$ = element of $F(X, ')$ corresponding to $w \in \mathcal{W}(X)$,

$\langle X, ' \mid \mathcal{R} \rangle$ = group generated by X subject to relators $\mathcal{R} \cup \{xx' \mid x \in X\}$ ($\mathcal{R} \subseteq \mathcal{W}(X)$) $= F(X, ')/\langle [\mathcal{R}]^F \rangle$, where $\langle [\mathcal{R}]^F \rangle$ is the smallest normal subgroup of $F(X, ')$ containing $\{[w] \mid w \in \mathcal{R}\}$.

If G is any group generated by a set S and if $w \in \mathcal{W}(S)$, let

$\overline{w}$ = element of G obtained by multiplying out the word w.

Assume, in addition, that S is closed under inversion and that s' denotes s^{-1} for $s \in S$. A word, $r \in \mathcal{W}(S)$, is called a relator in G if $\overline{r} = 1$. If $\mathcal{R} \subseteq \mathcal{W}(S)$ is a set of relators in G, then G is a homomorphic image of $\langle S, ' \mid \mathcal{R} \rangle$ (the natural homomorphism maps $\langle [\mathcal{R}]^F \rangle [w]$ to $\overline{w}$). If the homomorphism is an isomorphism, $\mathcal{R}$ is called a set of defining relators for G in terms of $(S, ')$.

3. Coset Tables and Interruptible Coset Enumeration. The Schreier-Todd-Coxeter-Sims algorithm makes use of a process called interruptible coset enumeration. Coset enumeration was outlined by Todd and Coxeter [10] in 1936. Let X be a finite set with an involutory map $x \longrightarrow x'$. Let $\mathcal{R}$ and $\mathcal{S}$ be two finite subsets of $\mathcal{W}(X)$.

A coset enumeration (or Todd-Coxeter) algorithm enumerates right cosets in

$G = \langle X, ' \mid \mathcal{R} \rangle$ of the subgroup $H = \langle S \rangle$. If the algorithm terminates, it finds $|G:H|$ and produces a table giving the action of X on the right cosets of H. An interruptible coset enumeration algorithm is one that may be stopped while in progress and later resumed from the point of interruption, possibly with additional relators or subgroup generators. Sims presents one such algorithm in [7] (in APL); the author has developed a fast interruptible coset enumeration algorithm (in FORTRAN and assembler).

Considerable effort has been devoted to proving that, under certain restrictions, coset enumeration algorithms must terminate provided $|G:H|$ is finite. These theorems will not be needed here because of the special way in which the STCS algorithm uses coset enumeration; in fact, all that will be needed is a formal definition of coset enumeration and several lemmas. The formal definition given here is taken from an approach developed by Sims (unpublished).

Sims defines a *coset table* to be a six-tuple $\mathcal{T} = (X, ', \Lambda, \iota, o, f)$, where X is a finite set with an involutory map $x \longrightarrow x'$ (if $w = x_1 x_2 \cdots x_l \in \mathcal{W}(X)$, w' will denote $x_l' x_{l-1}' \cdots x_1'$), and

(i) Λ is a finite set, $\iota \in \Lambda$, $o \in \Lambda$ ($\Lambda^{\#}$ will denote $\Lambda - \{o\}$),

(ii) f: $\Lambda \times X \longrightarrow \Lambda$ (if $w \in \mathcal{W}(X)$, $f(\lambda, w)$ is defined inductively to be $f(f(\lambda, w_1), x)$, where $w = w_1 x$),

(iii) $f(o, x) = o$ for all $x \in X$,

(iv) if $f(\lambda, x) = \mu \neq o$, then $f(\mu, x') = \lambda$,

(v) if $\lambda \in \Lambda^{\#}$, there exists $w \in \mathcal{W}(X)$ with $f(\iota, w) = \lambda$.

Note that, by (iv), the word w in (v) may be taken to be reduced; also w may be chosen with $\ell(w) < |\Lambda^{\#}|$. If $[w_1] = [w_2]$ in $F(X, ')$, then by (iv) $f(\lambda, w_1) = f(\lambda, w_2)$ if $f(\lambda, w_1), f(\lambda, w_2) \neq o$. Also by (iv), $f(\lambda, w) = \mu \neq o$ implies $f(\mu, w') = \lambda$ for any $w \in \mathcal{W}(X)$.

We think of $\Lambda^{\#}$ as being a set of right cosets of H in G, ι as being the coset $H1$, $f(\lambda, x)$ as being the image of coset λ under x, and $f(\lambda, x) = o$ as meaning that the image is currently undefined. Often ι and o are written as 1 and 0 respectively, and $f(\lambda, x)$ is written as λ^x.

A coset table $\mathcal{T}$ is *closed*, if $f(\lambda, x) \neq o$ whenever $\lambda \neq o$. In this case, a permutation $\hat{x}$ of $\Lambda^{\#}$ may be defined, for each $x \in X$, by $\hat{x}$: $\lambda \longrightarrow f(\lambda, x)$. Let $\hat{G}$ denote $\langle \hat{X} \rangle$, where $\hat{X} = \{\hat{x} \mid x \in X\}$.

Two coset tables $\mathcal{T}_1 = (X, ', \Lambda_1, \iota_1, o_1, f_1)$ and $\mathcal{T}_2 = (X, ', \Lambda_2, \iota_2, o_2, f_2)$ are *equivalent* if there is a 1 : 1 map j of Λ_1 onto Λ_2 such that $f_2(j(\lambda), x) = j(f_1(\lambda, x))$ for all λ and x, $j(\iota_1) = \iota_2$, and $j(o_1) = o_2$. Equivalent tables differ only in labeling of cosets.

An $(\mathcal{R}, S)$ *Todd-Coxeter algorithm* is an algorithm which transforms a coset table $\mathcal{T}_a$ to a table $\mathcal{T}_b$ by a series of steps $\mathcal{T}_a = \mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_N = \mathcal{T}_b$ such that, up to equivalence, $\mathcal{T}_{i+1}$ is obtained from $\mathcal{T}_i$ in one of the following ways:

(c.1) [definition of new coset]. For some $\lambda \in \Lambda_i^{\#}$ and $x \in X$ with $f_i(\lambda, x) = o_i$, and some $\mu \notin \Lambda_i$, set $\Lambda_{i+1} = \Lambda_i \cup \{\mu\}$, $\iota_{i+1} = \iota_i$, $o_{i+1} = o_i$, $f_{i+1}(\lambda, x) = \mu$, $f_{i+1}(\mu, x') = \lambda$, $f_{i+1}(\mu, y) = o_{i+1}$ if $y \neq x'$, and $f_{i+1}(\eta, z) = f_i(\eta, z)$ in all other cases.

(c.2) [deduction]. For some pair $(\lambda, w) \in (\Lambda_i^{\#} \times \mathcal{R}) \cup (\{\iota_i\} \times S)$ such that $w = w_1 x w_2$ $(x \in X)$, $\mu = f_i(\lambda, w_1) \neq o_i$, $\nu = f_i(\lambda, w_2') \neq o_i$, $f(\mu, x) = o_i$, and $f(\nu, x') = o_i$, set $\Lambda_{i+1} = \Lambda_i$, $\iota_{i+1} = \iota_i$, $o_{i+1} = o_i$, $f_{i+1}(\mu, x) = \nu$, $f_{i+1}(\nu, x') = \mu$, and $f_{i+1}(\eta, z) = f_i(\eta, z)$ otherwise.

(c.3) [equivalence]. For some pair $(\lambda, w) \in (\Lambda_i^{\#} \times \mathcal{R}) \cup (\{\iota_i\} \times S)$ such that $w = w_1 w_2$, $\mu = f_i(\lambda, w_1) \neq o_i$, $\nu = f_i(\lambda, w_2') \neq o_i$, and $\mu \neq \nu$, let $\sim$ be the smallest equivalence relation on Λ_i such that $\mu \sim \nu$ and that $f_i(\eta_1, x) \sim f_i(\eta_2, x)$ whenever $\eta_1 \sim \eta_2$ and $f_i(\eta_1, x), f_i(\eta_2, x) \neq o_i$. Let $\overline{\eta}$ denote the equivalence class of η. Set $\Lambda_{i+1} = \{\overline{\eta} \mid \eta \in \Lambda\}$, $o_{i+1} = \overline{\iota_i}$, $o_{i+1} = \overline{o_i}$, and $f_{i+1}(\overline{\eta}, x) = \overline{f_i(\kappa, x)}$ if there exists $\kappa \in \overline{\eta}$ with $f_i(\kappa, x) \neq o_i$ and $f_{i+1}(\overline{\eta}, x) = o_{i+1}$ otherwise. (Note f_{i+1} is well defined.)

If $|\Lambda_i^{\#}| \leqslant M$ for all i, then the algorithm is called an *M-coset* $(\mathcal{R}, S)$ *Todd-Coxeter* algorithm.

In a normal coset enumeration, one starts with a *trivial table* (i.e. $\Lambda = \{o, \iota\}$ and $f(\lambda, x) = o$ for all λ and x) and a fixed M; then one applies (c.1), (c.2), and (c.3) according to prescribed rules until one reaches a table in which all three operations are impossible; at that time, either $|\Lambda^{\#}| = M$ or $\mathcal{T}$ is closed. It can be shown that, in the latter case, $|G : H| = |\Lambda^{\#}|$ and $\hat{G}$ on $\Lambda^{\#}$ is permutation isomorphic to the representation of G on the right cosets of H (which need not be faithful). In interruptible coset enumeration, one may apply, in sequence, several Todd-Coxeter algorithms, each with a different set of relators and subgroup generators.

LEMMA 3.1. *If a Todd-Coxeter algorithm transforms* $\mathcal{T}_a$ *to* $\mathcal{T}_b$, *then*
(i) *if* $f_a(\iota_a, w) \neq o_a$, *then* $f_b(\iota_b, w) \neq o_b$,
(ii) *if* $f_a(\iota_a, w_1) = f_a(\iota_a, w_2) \neq o_a$, *then* $f_b(\iota_b, w_1) = f_b(\iota_b, w_2)$.

LEMMA 3.2. *There exists a bound d, depending only on M and* $|X|$, *such that, in at most d steps, any M-coset Todd-Coxeter algorithm terminates with a table to which* (c.1), (c.2), *and* (c.3) *cannot be applied. (The number of steps is defined to be the integer N in the preceding definition; note that the bound d is independent of* $\mathcal{R}$ *and* S.) *In that table, either* (i) $|\Lambda^{\#}| = M$ *or* (ii) $\mathcal{T}$ *is closed and* (c.3) *is impossible.*

Proof. If the ith step (i.e. $\mathcal{T}_{i-1} \longrightarrow \mathcal{T}_i$) involves (c.3), there exist $u, v \in \mathcal{W}(X)$ with $\ell(u), \ell(v) < M$, $f_j(\iota_j, u) \neq f_j(\iota_j, v)$ for $j < i$, and $f_j(\iota_j, u) = f_j(\iota_j, v)$ for $j \geqslant i$. If it involves (c.1) or (c.2), there exists $u \in \mathcal{W}(X)$ with $\ell(u) < M$, $f_j(\iota_j, u) = o_j$ for $j < i$, and $f_j(\iota_j, u) \neq o_j$ for $j \geqslant i$. Since the number of choices for u and v is bounded in terms of M and $|X|$, so is the number of applications of (c.1), (c.2), and (c.3). If $|\Lambda^{\#}| < M$ and $\mathcal{T}$ is not closed, then (c.1) can be performed; thus the last sentence of the lemma holds.

The next lemma is a slight variation of a well-known result.

LEMMA 3.3. *Let* Ω *be a finite set, S a set of permutations on* Ω *closed under inversion* (s' *denotes* s^{-1}), $L = \langle S \rangle$, $\beta \in \Omega$, *and* $\Delta = \beta^L$. *Let* $\mathcal{R} \subseteq \mathcal{W}(S)$ *be a set of relators in L, and let* $\mathcal{S} \subseteq \mathcal{W}(S)$ *be such that* $\overline{S} \subseteq L_\beta$. *If* $\mathcal{T}_a = (S, ', \Lambda_a, \iota_a, o_a, f_a)$ *is a coset table in which* $f_a(\iota_a, w_1) = f_a(\iota_a, w_2) \neq o_a$ *implies* $\beta^{\overline{w}_1} = \beta^{\overline{w}_2}$, *and if* $\mathcal{T}_b = (S, ', \Lambda_b, \iota_b, o_b, f_b)$ *is a table obtained from* $\mathcal{T}_a$ *by an* $(\mathcal{R}, \mathcal{S})$ *Todd-Coxeter algorithm,*

then

(i) $f_b(\iota_b, w_1) = f_b(\iota_b, w_2) \neq o_b$ *implies* $\beta^{\bar{w}_1} = \beta^{\bar{w}_2}$, *and* ψ: $f_b(\iota_b, w) \longrightarrow \beta^{\bar{w}}$ (*for* $f_b(\iota_b, w) \neq o_b$) *defines a map of* $\Lambda_b^{\#}$ *into* Δ,

(ii) *if* $\mathcal{T}_b$ *is closed,* ψ *maps* $\Lambda_b^{\#}$ *onto* Δ.

Sometimes, in a coset enumeration, the symbol $\mathcal{T}$ will be used for both initial and final tables as well as intermediate ones; in this case, $\mathcal{T}$ is implicitly a function of time.

Algorithms for performing (c.3) are well known (see, for example, [1, Section 3]); the author's version, which seems to be quite fast, is given below. It is assumed that $\Lambda^{\#}$ is ordered by $<$ such that ι comes first. While the algorithm is in progress, Υ will denote the set of cosets in Λ currently known to be equivalent to smaller cosets; initially Υ is empty; on termination, $\Lambda - \Upsilon$ will replace Λ. For $v \in \Upsilon$, $b(v)$ will be some coset in Λ smaller than v and equivalent to v. For $\lambda \in \Lambda$, the sequence defined by $\lambda_0 = \lambda$ and $\lambda_i = b(\lambda_{i-1})$, provided $\lambda_{i-1} \in \Upsilon$, must terminate with $\lambda_p \notin \Upsilon$ for some $p \geqslant 0$; define $b^*(\lambda) = \lambda_p$.[3] For $v \in \Upsilon$, $\delta(v)$ will be the next element added to Υ after v was added (or o if v is the element ζ_0 most recently added to Υ).

ALGORITHM 3.4—EQUIV($\mathcal{T}, \mu, \nu$). The coset table $\mathcal{T}$ is modified, as in (c.3), so as to make cosets μ and ν equivalent.

1. If $\mu = \nu$, algorithm terminates.
2. Set $\xi_1 = \min(\mu, \nu)$, $\xi_2 = \max(\mu, \nu)$, $\Upsilon = \varnothing$.
   Add ξ_2 to Υ; set $\delta(\xi_2) = o$, $\zeta_0 = \xi_2$.
   Set $b(\xi_2) = \xi_1$.
3. Set $\lambda = \xi_2$.
4. Set $\lambda^* = b^*(\lambda)$.
5. For each element x of X with $f(\lambda, x) \neq o$:[4]
   (a) Set $\tau = f(\lambda, x)$.
   (b) Set $\tau^* = b^*(\tau)$.
   (c) Set $f(\lambda, x) = o$, $f(\tau, x') = o$.
   (d) Set $\kappa = f(\lambda^*, x)$.
   (e) If $\kappa \neq o$:
      (i) Set $\kappa^* = b^*(\kappa)$.
      (ii) If $\kappa^* \neq \tau^*$:
         Set $\xi_1 = \min(\tau^*, \kappa^*)$, $\xi_2 = \max(\tau^*, \kappa^*)$.
         Add ξ_2 to Υ; set $\delta(\zeta_0) = \xi_2$, $\delta(\xi_2) = o$,

---

[3] Computation of $b^*(\lambda)$ is easy, provided a fast method to check if $\lambda \in \Upsilon$ is available. The exact method depends on the data structures used. Often a doubly linked list of the points of Λ is maintained. Assume that initially δ and b provide such a list, that is, that δ and $b = \delta^{-1}$ are permutations, each with just one cycle on Λ, such that $\delta(o) = \iota$. As the algorithm progresses, δ and b will continue to provide a doubly linked list of the points of $\Lambda - \Upsilon$, if one sets $\delta(b(\xi_2)) = \delta(\xi_2)$ and $b(\delta(\xi_2)) = b(\xi_2)$ whenever the algorithm specifies "add ξ_2 to Υ." For $v \in \Upsilon$, $\delta(v)$ and $b(v)$ are as set in the algorithm. Note that $\delta(b(\lambda)) = \lambda$ if and only if $\lambda \notin \Upsilon$.

[4] The elements of X may be transversed in any order; however, for a particular x, the condition $f(\lambda, x) \neq o$ must be checked after 5(a)–5(f) have been performed for any previous elements of X.

$\zeta_0 = \xi_2$.
Set $b(\xi_2) = \xi_1$.
If $\lambda^* = \xi_2$, set $\lambda^* = \xi_1$.

(f) If $\kappa = o$:

(i) Set $\eta = f(\tau^*, x')$.

(ii) If $\eta \neq o$:

(α) Set $\eta^* = b^*(\eta)$.

(β) If $\eta^* \neq \lambda^*$:

Set $\xi_1 = \min(\lambda^*, \eta^*)$, $\xi_2 = \max(\lambda^*, \eta^*)$.
Add ξ_2 to Υ; set $\beta(\zeta_0) = \xi_2$,
$\beta(\xi_2) = o$, $\zeta_0 = \xi_2$.
Set $b(\xi_2) = \xi_1$.
Set $\lambda^* = \xi_1$.

(iii) If $\eta = o$, set $f(\lambda^*, x) = \tau^*$ and $f(\tau^*, x') = \lambda^*$.

6. If $\lambda \neq \zeta_0$, set $\lambda = \beta(\lambda)$ and go to step 4.

7. Set $\Lambda = \Lambda - \Upsilon$; algorithm terminates.

When a pair σ_1 and σ_2 of equivalent cosets is discovered, equivalence of $b^*(\sigma_1)$ and $b^*(\sigma_2)$ is implied. If $b^*(\sigma_1)$ and $b^*(\sigma_2)$ are already equal, no action is required; otherwise $\xi_2 = \max(b^*(\sigma_1), b^*(\sigma_2))$ is added to Υ, and $b(\xi_2)$ is set to $\xi_1 = \min(b^*(\sigma_1), b^*(\sigma_2))$. Cosets in Υ are processed in the order in which they are added to Υ. Processing a coset λ in Υ involves transferring any information about λ to $\lambda^* = b^*(\lambda)$. Information is available for those elements x of X for which $\tau = \lambda^x$ is defined. Let τ^* denote $b^*(\tau)$. The definitions of λ^x and $\tau^{x'}$ are removed from the table. If both $(\lambda^*)^x$ and $(\tau^*)^{x'}$ are then undefined, $(\lambda^*)^x$ is set to τ^* and $(\tau^*)^{x'}$ is set to λ^*; otherwise a pair of equivalent cosets has been discovered–either $(\lambda^*)^x$ and τ^* or λ^* and $(\tau^*)^{x'}$.

4. Orbits and Schreier Vectors. Let $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ be a finite set with $0 \notin \Omega$, $S = \{s_1, s_2, \ldots, s_m\}$ be a set of permutations on Ω, and $G = \langle s_1, s_2, \ldots, s_m \rangle$. This section gives a computational solution to the problem of describing the G-orbits on Ω. A satisfactory description will consist of

(i) a list containing, for each G-orbit, a representative of the orbit and the length of the orbit,

(ii) a fast method for deciding if two points α_1 and α_2 of Ω are in the same G-orbit and, if so, for finding a word w in $W(S)$ with $\alpha_1^{\overline{w}} = \alpha_2$,

and perhaps of

(iii) a list of the points in each G-orbit.

To facilitate (ii) above, the concept of a Schreier vector will prove useful. Let $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_q)$ be an ordered subset of Ω. A *Γ-Schreier vector* for S on Ω is a function v: $\Omega \longrightarrow S \cup Z^-$ such that

(s.1) $v(\alpha) = 0$ if $\alpha \notin \Gamma^G$,

(s.2) $v(\alpha) = -i$ if $\alpha = \gamma_j$, $\gamma_j \notin \{\gamma_1, \ldots, \gamma_{j-1}\}^G$, and $\{\gamma_1, \ldots, \gamma_{j-1}\}^G$ consists of exactly $i - 1$ G-orbits,

(s.3) $\nu(\alpha) \in S$ if $\alpha \in \Gamma^G$ and α is not of the form of (s.2),

(s.4) for $\alpha \in \Gamma^G$, the sequence defined by

$$\alpha_1 = \alpha,$$
$$\alpha_{i+1} = \alpha_i^{\nu(\alpha_i)^{-1}} \text{ if } \nu(\alpha_i) \in S,$$

terminates with $\nu(\alpha_p) \in Z^-$ for some p.

A Γ-Schreier vector describes those orbits of G which contain some point of Γ. For each such orbit, the orbit representative ρ is the first point of the sequence $\gamma_1, \gamma_2, \ldots, \gamma_q$ lying in the orbit, and $\nu(\rho)$ is the negative of the orbit number. For any point α of Γ^G, the Schreier vector ν, together with S, contains all the information necessary to determine the orbit representative ρ of α^G and to compute a word $w \in \mathcal{W}(S)$ with $\rho^{\overline{w}} = \alpha$. Define $\pi_\nu(\alpha)$ to be the point α_p of (s.4) and $u_\nu(\alpha)$ to be the word $\nu(\alpha_{p-1})\nu(\alpha_{p-2}) \cdots \nu(\alpha_1)$; then ρ and w may be chosen to be $\pi_\nu(\alpha)$ and $u_\nu(\alpha)$ respectively.[5] Note that $\ell(u_\nu(\alpha)) < |\alpha^G| \leqslant n$, and for an orbit representative γ_i, $\{u_\nu(\alpha) \mid \alpha \in \gamma_i^G\}$ is a right Schreier system (see Hall [3, Section 7.2]), and $\{\overline{u}_\nu(\alpha) \mid \alpha \in \gamma_i^G\}$ is a set of right coset representatives for G_{γ_i} in G. Two points α_1 and α_2 of Γ^G are in the same G-orbit if and only if $\pi_\nu(\alpha_1) = \pi_\nu(\alpha_2)$, in which case $\alpha_1^{\overline{u}_\nu(\alpha_1)^{-1}\overline{u}_\nu(\alpha_2)} = \alpha_2$. Perhaps the most interesting cases are $\Gamma = \{\gamma_1\}$ (Schreier vector for one orbit) and $\Gamma = \Omega$ (complete Schreier vector).

Schreier vectors have been defined for permutation groups; they are also useful for working with coset tables. Assume, now, that S is any set with an involutory map $s \longrightarrow s'$ (with $S \cap Z^- = \varnothing$), Ω is any set, $\Gamma = (\gamma_1, \ldots, \gamma_q)$ is an ordered subset of Ω, and $T = (S, ', \Omega, \iota, o, f)$ is a coset table. A Γ-Schreier vector ν for T is a function $\nu\colon \Omega \longrightarrow S \cap Z^-$ such that

(t.1) $\nu(o) = 0$, $\nu(\gamma_1) = -1$,

(t.2) $\nu(\alpha) \in S$ if $\alpha \in \Omega^\# - \{\gamma_1\}$,

(t.3) for $\alpha \in \Omega^\#$, the sequence defined by

$$\alpha_1 = \alpha,$$
$$\alpha_{i+1} = f(\alpha_i, \nu(\alpha_i)') \text{ if } \nu(\alpha_i) \in S,$$

terminates with $\nu(\alpha_p) = -1$ for some p.

If (t.2) fails because $\nu(\alpha) = 0$ for some $\alpha \in \Omega^\#$, but (t.3) holds whenever $\nu(\alpha) \in S$, ν will be called a partial Γ-Schreier vector for T. Only γ_1 is specified in (t.1) because, by (iv) and (v) in the definition of a coset table, there is only one "orbit" on $\Omega^\#$. In fact, nothing is lost in assuming $\Gamma = \{\gamma_1\}$; usually $\gamma_1 = \iota$. For all α with $\nu(\alpha) \neq 0$, $u_\nu(\alpha)$ is defined to be the word $\nu(\alpha_{p-1})\nu(\alpha_{p-2}) \cdots \nu(\alpha_1)$; then $f(\gamma_1, u_\nu(\alpha)) = \alpha$. Always $\pi_\nu(\alpha) = \gamma_1$.

For future reference, the process of determining $\pi_\nu(\alpha)$ and $u_\nu(\alpha)$ from a Schreier vector ν, either in a permutation group or in a coset table, is formalized in the algorithm below.

---

[5] Note that S as well as ν is needed to compute $\pi_\nu(\alpha)$ and $u_\nu(\alpha)$. While constructing the Schreier vector ν, one can easily produce a second vector υ such that $\upsilon(\alpha) = \alpha^{\nu(\alpha)^{-1}}$ whenever $\nu(\alpha) \in S$. Then $\pi_\nu(\alpha)$ and $u_\nu(\alpha)$ can be computed from ν and υ without referring to S. This technique is especially useful for groups of relatively high degree, where it may be possible to hold ν and υ, but not S, in main storage.

ALGORITHM 4.1—COSETREP($\Omega, S, v, \alpha, \rho, w$). Initially Ω and S are as previously defined (in either permutation group or coset table case), v is a Γ-Schreier vector for S on Ω (or a Γ-Schreier vector or partial Γ-Schreier vector for T), and $\alpha \in \Omega$. The algorithm sets ρ to $\pi_v(\alpha)$ and w to $u_v(\alpha)$ if $v(\alpha) \neq 0$; if $v(\alpha) = 0$, it sets ρ to 0.

1. If $v(\alpha) = 0$, set $\rho = 0$; algorithm terminates.
2. Set $\rho = \alpha$, $w = e$.
3. If $v(\rho) \in Z^-$, algorithm terminates.
4. Set $w = v(\rho)w$, $\rho = \rho^{v(\rho)^{-1}}$ ($\rho = f(\rho, v(\rho)')$ in a coset table).
5. Go to step 3.

The next algorithm finds the orbit representatives and orbit lengths for G on Γ^G and constructs a Γ-Schreier vector. If S is ordered by $s_1 << s_2 << \cdots << s_m$, then $\mathcal{W}(S)$ may be ordered by $w_1 << w_2$, if $\ell(w_1) < \ell(w_2)$ or if $w_1 = w_0 s w^*$ and $w_2 = w_0 s^* w^*$ with $s, s^* \in S$ and $s << s^*$. The Γ-Schreier vector v constructed by the algorithm has the property that, for any $\alpha \in \Gamma^G$, $u_v(\alpha) = \min\{w \mid w \in \mathcal{W}(S), \pi_v(\alpha)^{\bar{w}} = \alpha\}$; also v is unique with this property. In particular, v is a minimal length Schreier vector, that is, $\ell(u_v(\alpha)) = \min\{\ell(w) \mid w \in \mathcal{W}(S), \pi_v(\alpha)^{\bar{w}} = \alpha\}$.

ALGORITHM 4.2—ORBIT($\Omega, S, \Gamma, n_0, \rho, l, \delta, v$). Initially, Ω, S, and Γ are as previously defined. The algorithm sets n_0 to the number of G-orbits on Γ^G, ρ_i and l_i to the representative and length for the ith orbit, $\delta_{l_1+\cdots+l_{i-1}+1}, \ldots, \delta_{l_1+\cdots+l_i}$ to a list of points in the ith orbit (with the orbit representative ρ_i coming first), and v to the Γ-Schreier vector for S on Ω having the minimality property defined above.

The algorithm may also be used to construct v when Ω and S are part of a coset table $T = (S, ', \Omega, \iota, o, f)$

1. Set $n_0 = 0$.
   Set $t = 0$, $t_0 = 0$.
   For $\beta = \omega_1, \omega_2, \ldots, \omega_n$, set $v(\beta) = 0$.
   Set $i = 1$.
2. If $v(\gamma_i) \neq 0$, go to step 7.
3. Set $n_0 = n_0 + 1$, $\rho_{n_0} = \gamma_i$, $l_{n_0} = 1$.
   Set $v(\gamma_i) = -n_0$.
   Set $t = t + 1$, $\delta_t = \gamma_i$.
4. Set $t_0 = t_0 + 1$, $\alpha = \delta_{t_0}$.
5. For $j = 1, 2, \ldots, m$:
   (a) Set $\beta = \alpha^{s_j}$.
   (b) If $v(\beta) = 0$ (and, in case of a coset table, if $\beta \neq o$):
      (i) Set $v(\beta) = s_j$.
      (ii) Set $l_{n_0} = l_{n_0} + 1$.
      (iii) Set $t = t + 1$, $\delta_t = \beta$.
6. If $t_0 < t$, go to step 4.
7. If $i < q$, set $i = i + 1$ and go to step 2;
   Otherwise algorithm terminates.

A point of Γ^G will be termed "found" when it first arises, either in steps 2–3 as an orbit representative γ_i or in step 5 as the image β of some point α previously found. A point will be termed "processed" when all the generators have been applied to it. Points are processed in the order in which they are found; this fact, together with the order in which generators are applied to a fixed point α in step 5, insures that the Schreier vector v has the minimality property discussed previously. While the algorithm is in progress, n_0 denotes the number of orbits found, t the number of points of Γ^G found, t_0 the number of the point currently being processed, $\delta_1, \ldots, \delta_{t_0}, \ldots, \delta_t$ the list of points found; and $v(\beta)$ is nonzero if and only if β has been found.

Step 1 initializes n_0, t, t_0, and v; a variable i is initialized to 1. Step 2 checks whether γ_i lies in a new orbit; if so, steps 3–6 construct the orbit of γ_i. Step 3 initializes the appropriate variables to correspond to an orbit of length 1 with representative γ_i. Step 4 sets α to the next point to be processed, and step 5 processes α; if new points β are found, they are added to the list of points to be processed, and the orbit length and Schreier vector are adjusted. If α is not the last point found, step 6 branches back to step 4 so that the next point may be processed; otherwise control passes to step 7, which increments i and branches back to step 2 unless there are no more points of Γ to be checked.

Algorithms 4.1 and 4.2 (with $\Gamma = \Omega$) satisfy the requirements (i), (ii), and (iii) for a satisfactory description of the G-orbits on Ω. Time requirements of Algorithm 4.2 are linear in mn when $\Gamma = \Omega$. This follows because step 1 is performed once and steps 2 through 7 at most n times; moreover, the time required to perform a given step once is proportional to n for step 1 and to m for step 5 and is constant otherwise. Assuming that the number of orbits of G on Γ^G is small compared to the degree n, the principal memory requirements of Algorithm 4.2 consist of $m + 2$ arrays of dimension n—one array for each permutation in S, one for δ, and one for v.

In Algorithm 4.2, the set S of generators need not be closed under inversion; however, to apply Algorithm 4.1 efficiently, inverses are necessary.

5. Bases and Strong Generating Sets. As mentioned in Section 1, Sims introduced the concepts of base and strong generating set to facilitate computation in large permutation groups. These concepts are discussed in [5] and [6]; only a summary of those results needed for the Schreier-Todd-Coxeter-Sims and random Schreier algorithms will be given here. An analogous concept, which I shall call strong defining relators, is included also.

The notation below will be used continually in Sections 5, 6, and 7.

DEFINITION 5.1.

$\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ is a finite set (with $0 \notin \Omega$),

$S = \{s_1, s_2, \ldots, s_m\}$ is a set of permutations on Ω,

$G = \langle S \rangle$,

$\mathcal{B} = (\beta_1, \beta_2, \ldots, \beta_k)$ is an ordered subset of (distinct elements) of Ω (possibly empty),

$G^{(i)} = G_{\beta_1\beta_2\cdots\beta_{i-1}}$, $1 \leqslant i \leqslant k + 1$,

$S^{(i)} = S \cap G_{\beta_1\beta_2\cdots\beta_{i-1}}$, $1 \leqslant i \leqslant k + 1$,

$H^{(i)} = \langle S^{(i)} \rangle$, $1 \leqslant i \leqslant k+1$,

$\Delta^{(i)} = \beta_i^{H^{(i)}}$, $1 \leqslant i \leqslant k$ (called the ith basic orbit),

$v^{(i)} = \{\beta_i\}$-Schreier vector for $S^{(i)}$ on Ω, $1 \leqslant i \leqslant k$,

$u^{(i)}(\delta) = u_{v^{(i)}}(\delta)$ for $\delta \in \Delta^{(i)}$, $1 \leqslant i \leqslant k$,

$U^{(i)} = \{u^{(i)}(\delta) \mid \delta \in \Delta^{(i)}\}$, $1 \leqslant i \leqslant k$ (a right Schreier system of right coset representatives for $H^{(i)}_{\beta_i}$ in $H^{(i)}$),

$n_i = |\Delta^{(i)}| = |U^{(i)}|$, $1 \leqslant i \leqslant k$,

$R = \{r_1, \ldots, r_p\}$ = subset of $W(S)$ consisting of relators in G,

$R^{(i)} = R \cap W(S^{(i)})$.

Although $\mathcal{B}$ and S need not be a base and strong generating set for G on Ω, it will be useful at times to assume the following.

HYPOTHESIS 5.2. (i) S is closed under inversion.

(ii) No element of S fixes $\mathcal{B}$ pointwise.

Should Hypothesis 5.2 fail to hold, it is necessary merely to delete the identity from S, if present, and then to expand $\mathcal{B}$ until (ii) holds and add inverses to S. Note that (ii) implies $S^{(k+1)} = \varnothing$ and $|H^{(k+1)}| = 1$.

Usually Ω, S, and $\mathcal{B}$ are known initially. Then $S^{(i)}$ may be determined by inspection. Using Algorithm 4.2, $\Delta^{(i)}$, n_i, and $v^{(i)}$ can be computed; then $u^{(i)}(\delta)$ may be found by Algorithm 4.1. In general, $G^{(i)}$ is more difficult to determine. Note that, by definition, $\mathcal{B}$ is a base and S is a strong generating set for G if and only if no element of S fixes $\mathcal{B}$ pointwise and $G^{(i)} = H^{(i)}$ for $i = 1, 2, \ldots, k+1$. This leads to several equivalent conditions for a base and strong generating set.

LEMMA 5.3. *If no element of S fixes $\mathcal{B}$ pointwise, the following are equivalent*:

(i) *$\mathcal{B}$ and S are a base and strong generating set for G.*

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

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

Proof. By definition, $G^{(i)}_{\beta_i} = G^{(i+1)}$. If (i) holds, then $H^{(i)}_{\beta_i} = G^{(i)}_{\beta_i} = G^{(i+1)} = H^{(i+1)}$, and (ii) holds. Always $G^{(1)} = H^{(1)}$. If (ii) holds, then $G^{(l)} = H^{(l)}$ implies $G^{(l+1)} = G^{(l)}_{\beta_l} = H^{(l)}_{\beta_l} = H^{(l+1)}$, and thus $G^{(i)} = H^{(i)}$ for all i; hence (i) holds. Equivalence of (ii) and (iii) follows from $H^{(i+1)} \subseteq H^{(i)}_{\beta_i}$ and $|H^{(i)} : H^{(i)}_{\beta_i}| = |\beta_i^{H^{(i)}}| = n_i$.

COROLLARY 5.4. *If $1 \leqslant i \leqslant k$ and $\mathcal{B}$ and $S^{(i+1)}$ are a base and strong generating set for $H^{(i+1)}$, the following are equivalent*:

(i) *$\mathcal{B}$ and $S^{(i)}$ are a base and strong generating set for $H^{(i)}$.*

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

(iii) $|H^{(i)} : H^{(i+1)}| = n_i$.

LEMMA 5.5. *Let $g \in S_\Omega$ with $\beta_l^g = \beta_l$ for $l = 1, 2, \ldots, p-1$. Then there is a unique integer j with $p \leqslant j \leqslant k+1$, unique words $u_p, u_{p+1}, \ldots, u_{j-1}$ with $u_q \in U^{(q)}$ $(q = p, \ldots, j-1)$, and a unique element h in S_Ω fixing $\beta_1, \ldots, \beta_{p-1}, \ldots, \beta_{j-1}$ such that $g = h\bar{u}_{j-1}\bar{u}_{j-2} \cdots \bar{u}_p$ and either*

(1) *$j \leqslant k$ and $\beta_j^h \notin \Delta^{(j)}$ (hence $h \notin H^{(j)}$),*

(2) $j = k + 1$ *and* $h \neq 1$, *or*

(3) $j = k + 1$ *and* $h = 1$.

If $\mathcal{B}$ *and* S *are a base and strong generating set for* G, *then* $g \in G$ *if and only if* (3) *holds.*

The next algorithm determines j and constructs $u_p, u_{p+1}, \ldots, u_{j-1}$ and h. The u_l's are chosen in the order listed; at each stage only one choice is possible for $u^{(l)}$; thus uniqueness holds.

ALGORITHM 5.6–ELEMENT $(\Omega, S, \mathcal{B}, \{v^{(l)}\}, p, g, j, \{u_l\}, h)$. Finds j, $\{u_l\}_{l=p}^{j-1}$, and h in Lemma 5.5.

1. Set $j = p$, $h = g$.[6]
2. If $j > k$, the algorithm terminates ((2) holds if $h \neq 1$ and (3) holds otherwise).
3. If $v^{(j)}(\beta_j^h) = 0$, the algorithm terminates ((1) holds).
4. Apply Algorithm 4.1–COSETREP $(\Omega, S^{(j)}, v^{(j)}, \beta_j^h, \rho, u_j)$ to find $u_j = u^{(j)}(\beta_j^h)$; set $h = hu_j^{-1}$.
5. Set $j = j + 1$ and go to step 2.

Algorithm 5.6 provides an effective means for testing whether $g \in U^{(k)}U^{(k-1)} \cdots U^{(1)}$ and, in the case of a base and strong generating set, whether $g \in G$. By uniqueness in Lemma 5.5, $|U^{(k)}U^{(k-1)} \cdots U^{(1)}| = \Pi_{i=1}^k n_i$. The next lemma will be used in the random Schreier method.

LEMMA 5.7. $|G|$ *is divisible by* $|U^{(k)}U^{(k-1)} \cdots U^{(1)}| = \Pi_{i=1}^k n_i$; $G = U^{(k)}U^{(k-1)} \cdots U^{(1)}$ *and* $|G| = \Pi_{i=1}^k n_i$ *if and only if* $\mathcal{B}$ *and* S *are a base and strong generating set for* G.

Proof. As $|H^{(i)} : H^{(i+1)}| = |H^{(i)} : H_{\beta_i}^{(i)}| \cdot |H_{\beta_i}^{(i)} : H^{(i+1)}| = n_i |H_{\beta_i}^{(i)} : H^{(i+1)}|$,
$|G| = (\Pi_{i=1}^k |H^{(i)} : H^{(i+1)}|) \cdot |H^{(k+1)}| = (\Pi_{i=1}^k n_i) \cdot (\Pi_{i=1}^k |H_{\beta_i}^{(i)} : H^{(i+1)}|) \cdot |H^{(k+1)}|$.
Hence the first assertion of the lemma holds, and $|G| = \Pi_{i=1}^k n_i$ if and only if $|H^{(k+1)}| = 1$ and $H_{\beta_i}^{(i)} = H^{(i+1)}$ for all i, which by Lemma 5.3 is true exactly when $\mathcal{B}$ is a base and S is a strong generating set.

In the Introduction, three questions were posed about the group G; these questions may now be answered quite easily provided $\mathcal{B}$ and S are a base and strong generating set for G on Ω. To find $|G|$, we compute n_i $(i = 1, 2, \ldots, k)$ by Algorithm 4.2 and apply the formula $|G| = \Pi_{i=1}^k n_i$. To test whether $g \in G$, we apply Algorithm 5.6 and check whether alternative (3) of Lemma 5.5 holds at termination; if so, $g \in G$ and $u_k u_{k-1} \cdots u_1 \in W(S)$ with $g = \bar{u}_k \bar{u}_{k-1} \cdots \bar{u}_1$. To choose a random element of $G = U^{(k)}U^{(k-1)} \cdots U^{(1)}$, we need merely choose random elements of each $U^{(i)}$ and take their product (by uniqueness in Lemma 5.5). Assuming that we have a function π such that $\pi(t)$ returns a random integer between 1 and t, we compute $\Delta^{(i)} = \{\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n_i}\}$ by Algorithm 4.2 and choose

$$\bar{u}^{(k)}(\delta_{k,\pi(n_k)})\bar{u}^{(k-1)}(\delta_{k-1,\pi(n_{k-1})}) \cdots \bar{u}^{(1)}(\delta_{1,\pi(n_1)})$$

as the random element of G (using Algorithm 4.1 to find the $u^{(l)}$'s).

[6] To carry out the algorithm, only the images of $\mathcal{B}$ under h are needed; however, to decide whether $h = 1$ in step 5, the images of all of Ω are needed, unless a smaller base for $\langle G, g \rangle$ is known.

Assume that $\mathcal{B}$ and S are a base and strong generating set for G on Ω and that S is closed under inversion (s' will denote s^{-1}). In general, it is not easy to check whether the set R of relators forms a set of defining relators for G in terms of $(S, ')$. Even if an $(R, S^{(2)})$ Todd-Coxeter algorithm applied to a trivial table $(S, ', \Lambda, \ldots)$ terminates with a closed table in which $|\Lambda^{\#}| = n_1$, we learn merely that the kernel of the natural homomorphism of $\langle S, ' \mid R \rangle$ onto G is contained in the subgroup generated by $S^{(2)}$. To alleviate this difficulty, R will be called a *strong set of defining relators* for G relative to $\mathcal{B}$ and $(S, ')$, if $R^{(i)}$ is a set of defining relators for $H^{(i)}$ in terms of $(S^{(i)}, ')$ for $i = 1, 2, \ldots, k$.

LEMMA 5.8. *If* $|\langle S^{(i)}, ' \mid R^{(i)} \rangle : \langle S^{(i+1)} \rangle| = n_i$ *for* $i = 1, 2, \ldots, k$, *then* R *is a strong set of defining relators for* G *relative to* $\mathcal{B}$ *and* $(S, ')$.

Proof. Let $T^{(i+1)}$ denote the subgroup of $\langle S^{(i)}, ' \mid R^{(i)} \rangle$ generated by $S^{(i+1)}$ (to be distinguished from $H^{(i+1)}$, the permutation group on Ω generated by $S^{(i+1)}$). Then $T^{(i+1)}$ is a homomorphic image of $\langle S^{(i+1)}, ' \mid R^{(i+1)} \rangle$ as the relators $R^{(i+1)}$ hold in $T^{(i+1)}$. Thus $|\langle S^{(i)}, ' \mid R^{(i)} \rangle| = |\langle S^{(i)}, ' \mid R^{(i)} \rangle : T^{(i+1)}| \cdot |T^{(i+1)}| \leqslant n_i |\langle S^{(i+1)}, ' \mid R^{(i+1)} \rangle|$; by repeated application of this formula, $|\langle S^{(i)}, ' \mid R^{(i)} \rangle| \leqslant n_i n_{i+1} \cdots n_k = |H^{(i)}|$, and the natural homomorphism of $\langle S^{(i)}, ' \mid R^{(i)} \rangle$ onto $H^{(i)}$ must be an isomorphism.

Note that the hypothesis of Lemma 5.8 will hold if, for $i = 1, 2, \ldots, k$, an $(R^{(i)}, S^{(i+1)})$ Todd-Coxeter algorithm applied to a trivial coset table $(S^{(i)}, ', \Lambda, \ldots)$ terminates with a closed table in which $|\Lambda^{\#}| = n_i$.

6. The Schreier-Todd-Coxeter-Sims Method. This section describes the author's version of the Schreier-Todd-Coxeter-Sims method and points out similarities and differences between it and the Schreier-Sims method. Both methods are motivated by a theorem of Schreier (see [3, Lemma 7.2.2]).

LEMMA 6.1 (SCHREIER). *Let* G *be a group generated by a set* S, *let* H *be a subgroup of* G, *and let* U *be a set of right coset representatives for* H *in* G. *For* $g \in G$, *let* $\bar{g}$ *denote the representative in* U *for the right coset* Hg. *Then* H *is generated by* $\{us\bar{u}\bar{s}^{-1} \mid u \in U, s \in S\}$.

The elements $us\bar{u}\bar{s}^{-1}$ are called *Schreier generators* for H. If G is a permutation group on Ω and $\beta \in \Omega$, then a set of Schreier generators for G may be obtained by choosing $\{\bar{u}_v(\alpha)s\bar{u}_v(\alpha^s)^{-1} \mid \alpha \in \beta^G, s \in S\}$, where v is a $\{\beta\}$-Schreier vector for S on Ω.

Note that this set may be computed using Algorithms 4.1 and 4.2. In principle, this solves the base and strong generating set problem (*) of Section 1, as a generating set for each group in the sequence $G, G_{\beta_1}, \ldots, G_{\beta_1\beta_2\cdots\beta_k}$ may be computed from a generating set for the previous group by taking Schreier generators, and $\beta_1, \beta_2, \ldots, \beta_k$ may be extended, if necessary, until the last group in the chain is trivial. However, the strong generating set obtained is far too large to be of practical value. In the Schreier-Sims and Schreier-Todd-Coxeter-Sims methods, this crude approach is modified so as to keep the size of the strong generating set relatively small.

With notation as in Definition 5.1, both methods involve expanding $\mathcal{B}$ and S so that Hypothesis 5.2 holds and then checking, for $i = k, k-1, \ldots, 2, 1$ (in that order),

that $\mathcal{B}$ and $S^{(i)}$ are a base and strong generating set for $H^{(i)}$. At the time that $\mathcal{B}$ and $S^{(i)}$ are checked, it is known that $\mathcal{B}$ and $S^{(i+1)}$ are a base and strong generating set for $H^{(i+1)}$; thus by Corollary 5.4, it is necessary merely to show that $H^{(i)}_{\beta_i} = H^{(i+1)}$ (or equivalently $|H^{(i)} : H^{(i+1)}| = n_i$); containment in $H^{(i+1)}$ may be checked using Algorithm 5.6. Should $H^{(i)}_{\beta_i} \neq H^{(i+1)}$, we obtain an element g of $H^{(i)}_{\beta_i} - H^{(i+1)}$. At this stage, g could be added to S, thereby enlarging $H^{(i+1)}$; however, a better approach is available. In determining that $g \notin H^{(i+1)}$ using Algorithm 5.6, g is written as $h\bar{u}_{j-1}\bar{u}_{j-2} \cdots \bar{u}_{i+1}$ with $j \geq i+1$, h fixing $\beta_1, \ldots, \beta_{j-1}$, $u^{(l)} \in U^{(l)}$ for all l, and either (1) $j \leq k$ and $\beta_j^h \notin \Delta^{(j)}$ or (2) $j = k+1$ and $h \neq 1$. In either case, $h \in G^{(j)}$ but $h \notin H^{(j)}$. We add h to S, thereby enlarging each of the groups $H^{(i+1)}$, $H^{(i+2)}, \ldots, H^{(j)}$. If $j = k+1$, a new point, moved by h, must be added to $\mathcal{B}$ in order for Hypothesis 5.2 to continue to hold. As $h \notin S^{(l)}$ for any $l > j$, the previous checks of $\mathcal{B}$ and $S^{(l)}$ remain valid for all $l > j$; hence i is reset to j.

A general outline for the Schreier-Todd-Coxeter-Sims method follows. If references to the set R of relators and the function M are deleted, the same outline is applicable for the Schreier-Sims method. The two methods differ in the technique for checking $H^{(i)}_{\beta_i} = H^{(i+1)}$, details of which are given later.

ALGORITHM 6.2–SCHREIER-TODD-COXETER-SIMS. The notation of Definition 5.1 applies. In addition, $M: Z^+ \longrightarrow Z^+$ is a function with $M(x) > x$ for all x. The identity is deleted from S if present; then the sets $\mathcal{B}$, S, and R are extended if necessary so that, upon termination, $\mathcal{B}$ is a base for G on Ω, S is a strong generating set for G relative to $\mathcal{B}$, and R is a strong set of defining relators for G relative to $\mathcal{B}$ and $(S, ')$. (Elements added to S lie in the original group G, so G remains unchanged.)

1. Delete the identity from S if present.
   Set $m_0 = m$.
   For $l = 1, 2, \ldots, m_0$:
   (a) If s_l fixes $\beta_1, \beta_2, \ldots, \beta_k$, set $k = k+1$, $\beta_k =$ any point moved by s_l.
   (b) If $s_l^{-1} \notin S$, set $m = m+1$, $s_m = s_l^{-1}$.
   (This insures that Hypothesis 5.2 holds.)
2. Set $i = k$.
   (At any time, $\mathcal{B}$ and $S^{(i+1)}$ will be known to be a base and strong generating set for $H^{(i+1)}$.)
3. For $l = 1, 2, \ldots, k$, set $\eta_l = 0$.
   ($\eta_l = 1$ will indicate that a valid $\{\beta_i\}$-Schreier vector $v^{(i)}$ for $S^{(i)}$ on Ω is known.)
4. If $\eta_i = 0$:
   (a) Apply Algorithm 4.2–ORBIT$(\Omega, S^{(i)}, \{\beta_i\}, n_0, \rho, n_i, \delta, v^{(i)})$.
   (b) Set $\eta_i = 1$.
   (This constructs the ith basic orbit $\delta = \{\delta_t\}_{t=1}^{n_i}$ and produces a $\{\beta_i\}$-Schreier vector $v^{(i)}$ for $S^{(i)}$.)
5. Check if $H^{(i)}_{\beta_i} = H^{(i+1)}$ (details later).

If so:

(a) Set $i = i - 1$.

(b) If $i = 0$, algorithm terminates; otherwise go to step 4.

If not:

(c) Find $w \in W(S^{(i)})$, such that $\bar{w} \in H^{(i)}_{\beta_i}$, but $\bar{w} \notin H^{(i+1)}$.

(d) Write $\bar{w} = h\bar{u}_{j-1}\bar{u}_{j-2} \cdots \bar{u}_{i+1}$ with $j \leqslant k+1$, h fixing $\beta_1, \beta_2, \ldots, \beta_{j-1}$, $u_l \in U^{(l)}$ for all l, and either

(i) $j \leqslant k$, $\beta_j^h \notin \Delta^{(j)}$, or

(ii) $j = k+1$, $h \neq 1$.

6. If $j = k+1$, set $k = k+1$, $\beta_k =$ any point moved by h.
Set $m = m+1$, $s_m = h$.
If $h^2 \neq 1$, set $m = m+1$, $s_m = h^{-1}$.
Set $p = p+1$, $r_p = wu_{i+1}^{-1} \cdots u_{j-1}^{-1}$.
For $l = i+1, i+2, \ldots, j$, set $\eta_l = 0$.

(S, R, and, if necessary, $\mathcal{B}$ are extended; the relator r_p expresses the new generator as a word in the previous generators; η_l is set to zero when the new generator lies in $S^{(l)}$.)

7. Set $i = j$.
Go to step 4.

The purpose of the η's is to avoid unnecessary reconstruction of the Schreier vectors in step 4. It is possible to delete all references to the η's, in which case step 4(a) is performed unconditionally. This will produce shorter relators, though with more occurrences of the added generators. Of course, when STCS is used merely to verify strong generation, it makes no difference.

PROPOSITION 6.3. *Assuming that step* 5 *terminates, Algorithm* 6.2 *terminates. Upon termination,* $\mathcal{B}$ *is a base for* G *on* Ω, *and* S *is a strong generating set for* G *relative to* $\mathcal{B}$.

Proof. Step 6 increases $|H^{(i)}|$ for some i with $1 \leqslant i < n$; hence it can be performed only finitely many times. Eventually, i is decremented each time through the loop beginning with step 4; thus i must reach 0, and the algorithm terminates.

At any time, $\mathcal{B}$ and $S^{(i+1)}$ form a base and strong generating set for $H^{(i+1)}$. This holds initially as $i = k$ and $H^{(k+1)} = 1$ (by Hypothesis 5.2). In step 5, i is decremented only after verifying that $H^{(i)}_{\beta_i} = H^{(i+1)}$, in which case Corollary 5.4 applies. Step 6 enlarges $H^{(l)}$ only for $l = i+1, i+2, \ldots, j$; so resetting i to j keeps the result valid.

Thus, when i reaches 0, $\mathcal{B}$ and $S = S^{(1)}$ are a base and strong generating set for $G = H^{(1)}$.

In the Schreier-Sims method, the check that $H^{(i)}_{\beta_i} = H^{(i+1)}$ is performed by testing whether each Schreier generator $\bar{w} = \bar{u}^{(i)}(\delta)s\bar{u}^{(i)}(\delta^s)^{-1}$ ($\delta \in \Delta^{(i)}$, $s \in S^{(i)}$)[7] of $H^{(i)}_{\beta_i}$

[7] Redundant generators for $H^{(i)}$ need not be included. If $s \in S^{(i)}$, only one of s and s^{-1} need be included; also, if s was added to S in step 6, s is redundant for $H^{(1)}, \ldots, H^{(i^*)}$, where i^* denotes the value of i in Algorithm 6.2 at the time that s was added.

lies in $H^{(i+1)}$. The test is done by writing $\bar{w}$ in the form $h\bar{u}_{j-1} \cdots \bar{u}_{i+1}$ using Algorithm 5.6. If $j = k + 1$ and $h = 1$ for every $\bar{w}$, then $H^{(i)}_{\beta_i} = H^{(i+1)}$; otherwise we obtain the word w required in step 5(c) and the factorization $h\bar{u}_{j-1} \cdots \bar{u}_{i+1}$ of step 5(d). Unfortunately, checking whether $h = 1$ requires applying the word $u^{(i)}(\delta)su^{(i)}(\delta^s)^{-1}u_{i+1}^{-1} \cdots u_k^{-1}$ to each of the n points of Ω;[8] together with the fact that there are $1 + n_i(|S^{(i)}| - 1)$ nontrivial Schreier generators, this accounts for the slowness of the Schreier-Sims method for groups of high degree.

We may view the Schreier-Sims method as consisting of verification of each relator $u^{(i)}(\delta)su^{(i)}(\delta^s)^{-1}u_{i+1}^{-1} \cdots u_k^{-1}$ ($\delta \in \Delta^{(i)}$, $s \in S^{(i)}$). These relators suffice to prove $H^{(i)}_{\beta_i} = H^{(i+1)}$; often, however, they are highly redundant. The Schreier-Todd-Coxeter-Sims method consists of verifying only selected relators. A list R of verified relators in S is maintained; $R^{(i)}$ will denote $R \cap W(S^{(i)})$. After a new relator has been selected and verified, it is added to R, and the group generated by $S^{(i)}$ (or, more precisely, by a set of abstract symbols in one-to-one correspondence with $S^{(i)}$) subject to relators $R^{(i)}$ is enumerated on the cosets of the subgroup generated by $S^{(i+1)}$, allowing $M(n_i) > n_i$ cosets to be defined. Should the coset table close with n_i cosets, then $|H^{(i)} : H^{(i+1)}| = n_i$ and no further relators need be verified (Corollary 5.4); otherwise words w_1 and w_2 in $W(S^{(i)})$ of minimal length are found such that 1^{w_1} and 1^{w_2} are defined and unequal in the coset table but $\beta_i^{\bar{w}_1} = \beta_i^{\bar{w}_2}$ on Ω, w is set to $w_1w_2^{-1}$, and $\bar{w}$ is written in the form $h\bar{u}_{j-1} \cdots \bar{u}_{i+1}$. If $j \leqslant k$ or $h \neq 1$, then $H^{(i)}_{\beta_i} \neq H^{(i+1)}$, and the quantities $w, h, u_{j-1}, \ldots, u_{i+1}$ required in steps 5(c) and 5(d) have been found; otherwise the relator $wu_{i+1}^{-1} \cdots u_k^{-1}$ is added to R, and the coset enumeration is resumed. Immediately upon resumption, the new relator is traced from coset 1, yielding equivalence of cosets 1^{w_1} and 1^{w_2}; thus new cosets can be introduced, even if $M(n_i)$ cosets were defined at the time of interruption. It will be shown that this process must terminate.

In the STCS method, step 5 of Algorithm 6.2 is performed as follows.

5.1. Initialize $T = (S^{(i)}, ', \Lambda, 1, 0, f)$ to a trivial coset table by setting $\Lambda = \{0, 1\}$ and $f(\lambda, s) = 0$ for $\lambda \in \{0, 1\}$ and $s \in S^{(i)}$.

5.2. Apply an $M(n_i)$-coset $(R^{(i)}, S^{(i+1)})$ Todd-Coxeter algorithm to T; at termination, one of the following must hold:

(i) $|\Lambda^{\#}| > n_i$,

(ii) $|\Lambda^{\#}| = n_i$ and T is closed.

5.3. If $|\Lambda^{\#}| = n_i$, then $H^{(i)}_{\beta_i} = H^{(i+1)}$; proceed with step 5(a) in the outline of Algorithm 6.2.

5.4. Find $w_1, w_2 \in W(S^{(i)})$ with $f(1, w_1) \neq f(1, w_2)$ in Λ, $\beta_i^{\bar{w}_1} = \beta_i^{\bar{w}_2}$ on Ω, $\ell(w_1) \geqslant \ell(w_2)$, and $\ell(w_1)$ minimal with respect to these properties (details will follow).

---

[8] If, however, a small base for G on Ω is already known, then the word need be applied only to the base points. This situation arises when we have a group E on Ω with known base and strong generating set and are trying to determine the subgroup generated by some given subset of E. The relative efficiency of the Schreier-Sims method in this special situation merits further investigation.

5.5. Apply Algorithm 5.6–ELEMENT($\Omega, S^{(i+1)}, \mathcal{B}, \{v^{(l)}\}, i+1, \bar{w}_1\bar{w}_2^{-1}, \{u^{(l)}\}, h$)– to write $\bar{w}_1\bar{w}_2^{-1} = h\bar{u}_{j-1}\cdots\bar{u}_{i+1}$, with either

(i) $j \leqslant k$, $\beta_j^h \notin \Delta^{(j)}$,

(ii) $j = k+1$, $h \neq 1$, or

(iii) $j = k+1$, $h = 1$.

5.6. If case (iii) above holds:

(a) Set $p = p+1$, $r_p = w_1 w_2^{-1} u_{i+1}^{-1}\cdots u_k^{-1}$.

(b) Apply Algorithm 3.4–EQUIV($\mathcal{T}, f(1, w_1), f(1, w_2)$).

(c) Go to step 5.2.

5.7. Step 5 is complete; $H_{\beta_i}^{(i)} \neq H^{(i+1)}$, and the quantities $w = w_1 w_2^{-1}$, h, $u_{j-1}, \ldots, u_{i+1}$ of steps 5(c) and 5(d) have been found.

To complete Algorithm 6.2, a procedure for finding the words w_1 and w_2 in step 5.4 is needed. The procedure described below is essentially that given by Sims in [7], modified to eliminate one array. A well-defined map of $\Lambda^{\#}$ into $\Delta^{(i)}$ may be obtained by $f(1, w) \longrightarrow \beta_i^{\bar{w}}$, whenever $w \in \mathcal{W}(S^{(i)})$ and $f(1, w) \neq 0$ (Lemma 3.3). This map cannot be one-to-one as $|\Lambda^{\#}| > |\Delta^{(i)}|$; hence w_1 and w_2 exist. To find them, we initialize a map $\theta\colon \Delta^{(i)} \longrightarrow \Lambda$ by setting $\theta(\beta_i) = 1$ and $\theta(\gamma) = 0$ for all $\gamma \neq \beta_i$, and we begin to construct a $\{1\}$-Schreier vector v for $S^{(i)}$ on Λ, as in Algorithm 4.2. When a new coset $\tilde{\gamma}$ is found by applying generator s_l to some previous coset $\tilde{\alpha} = \theta(\alpha)$, then the corresponding point γ of $\Delta^{(i)}$ is found by applying s_l to α. If $\theta(\gamma) = 0$, then $\theta(\gamma)$ is set to $\tilde{\gamma}$, indicating that γ corresponds to coset $\tilde{\gamma}$ just found. If $\theta(\gamma) \neq 0$, then $\tilde{\gamma}$ and $\theta(\gamma)$ are two cosets with the same corresponding point γ, and $w_1 = u_v(\tilde{\gamma})$ and $w_2 = u_v(\theta(\gamma))$ are the words we are looking for. The fact that $\ell(w_1) \geqslant \ell(w_2)$ and the minimality property of $\ell(w_1)$ follow from the fact that new cosets $\tilde{\gamma}$ are found with $\ell(u_v(\tilde{\gamma}))$ in increasing order.

The details for step 5.4 are as follows.

5.4.1. Set $t_0 = 0$,
$t = 1$, $\delta_1 = \beta_i$,
$\theta(\beta_i) = 1$, $\theta(\gamma) = 0$ for $\gamma \in \Delta^{(i)} - \{\beta_i\}$,
$v(1) = -1$, $v(\tilde{\gamma}) = 0$ for $\tilde{\gamma} \in \Lambda^{\#} - \{1\}$.

5.4.2. Set $t_0 = t_0 + 1$, $\alpha = \delta_{t_0}$, $\tilde{\alpha} = \theta(\alpha)$.[9]

5.4.3. For $l = 1, 2, \ldots, m$:

(a) Set $\tilde{\gamma} = f(\tilde{\alpha}, s_l)$.

(b) If $\tilde{\gamma} \neq 0$ and $v(\tilde{\gamma}) = 0$:

(i) Set $v(\tilde{\gamma}) = s_l$.

(ii) Set $\gamma = \alpha^{s_l}$.

(iii) If $\theta(\gamma) \neq 0$, go to step 5.4.5.

(iv) Set $\theta(\gamma) = \tilde{\gamma}$.

(v) Set $t = t+1$, $\delta_t = \gamma$.

[9] Note that, even after t_0 has been incremented, $t_0 \leqslant t$; otherwise every coset found would have been processed, and by condition (v) in the definition of a coset table, every coset would have been found, that is, $|\Lambda^{\#}| = t$. But $t \leqslant n_i$, as t is incremented by 1 only when a new point of $\Delta^{(i)}$ has been found, and $n_i < |\Lambda^{\#}|$.

5.4.4. Go to step 5.4.2.

5.4.5. Apply Algorithm 4.1–COSETREP$(\Lambda, S^{(i)}, v, \tilde{\gamma}, \rho, w_1)$.
Apply Algorithm 4.1–COSETREP$(\Lambda, S^{(i)}, v, \theta(\gamma), \rho, w_2)$.

PROPOSITION 6.4. *Step* 5 *of Algorithm* 6.2 *terminates.*

Proof. By Lemma 3.2, the Todd-Coxeter algorithm in step 5.2 cannot continue indefinitely without reaching a table in which $|\Lambda^{\#}| = M(n_i) > n_i$, or in which T is closed and (c.3) is impossible. In the latter case, $|\Lambda^{\#}| \geqslant n_i$ by Lemma 3.3 (since the relators $R^{(i)}$ hold in $H^{(i)}$). Thus alternative (i) or (ii) in step 5.2 must hold eventually.

Step 5 contains two loops–an outer loop extending from 5.2 through 5.6 and an inner one from 5.4.2 through 5.4.4. The inner loop is clearly finite, since t_0 is incremented each time through and since $t_0 \leqslant t \leqslant n_i$. For each pass through the outer loop, there must exist words w_1 and w_2 in $W(S^{(i)})$, such that $\ell(w_2) \leqslant \ell(w_1) < M(n_i)$ (by minimality), $f(1, w_1) \neq f(1, w_2)$ before the pass, and $f(1, w_1) = f(1, w_2)$ afterward. As there are only finitely many choices for the pair $\{w_1, w_2\}$, the outer loop must be finite.

PROPOSITION 6.5. *When Algorithm* 6.2 *terminates, R is a strong set of defining relators for G relative to* $\mathcal{B}$ *and* $(S, ')$.

Proof. Immediate from Lemma 5.8.

7. **The Random Schreier Method.** The random Schreier method provides a fast means for adding elements to $\mathcal{B}$ and S (notation of Definition 5.1 applies here). Each element added expands $U^{(k)}U^{(k-1)} \cdots U^{(1)}$, but upon termination $\mathcal{B}$ and S need not be a base and strong generating set for G on Ω. An input parameter N is supplied to the algorithm; increasing N improves the chance of obtaining a base and strong generating set but also adds to the running time.

Assume that $\mathcal{B}$ and S have been extended, if necessary, so that Hypothesis 5.2 holds. Let g be any element of G. Algorithm 5.6 provides a fast method for testing if $g \in U^{(k)}U^{(k-1)} \cdots U^{(1)}$. If not, $g = h\bar{u}_{j-1}\bar{u}_{j-2} \cdots \bar{u}_1$ with $j \leqslant k+1$, $\beta_l^h = \beta_l$ for $l \leqslant j-1$, $u_l \in U^{(l)}$ for all l, and either (1) $j \leqslant k$ and $\beta_j^h \notin \Delta^{(j)}$ or (2) $j = k+1$ and $h \neq 1$. Just as in the Schreier-Sims and STCS algorithms, h is added to S and, in case (2), a point moved by h is added to $\mathcal{B}$; thus $|U^{(k)}U^{(k-1)} \cdots U^{(1)}|$ is increased. If $g \in U^{(k)}U^{(k-1)} \cdots U^{(1)}$, no information is obtained. However, if a number of more or less random elements of G are tested and all turn out to lie in $U^{(k)}U^{(k-1)} \cdots U^{(1)}$, we suspect that $U^{(k)}U^{(k-1)} \cdots U^{(1)} = G$ and, by Lemma 5.7, that $\mathcal{B}$ and S are a base and strong generating set for G; for if not $|U^{(k)}U^{(k-1)} \cdots U^{(1)}| \leqslant \frac{1}{2}|G|$ (Lemma 5.7), and the probability that t statistically random elements of G all lie in $U^{(k)}U^{(k-1)} \cdots U^{(1)}$ is at most 2^{-t}. The random Schreier method chooses elements of G, ideally random in G, until N consecutive elements lie in $U^{(k)}U^{(k-1)} \cdots U^{(1)}$; then it terminates.

In Section 5, a fast method for selecting elements at random from G was given; however, this method required that some base and strong generating set for G (not necessarily $\mathcal{B}$ and S) already be known. Thus, in general, the algorithm must proceed

with elements not statistically random in G, and the probability bound given above does not apply; nevertheless, the algorithm seems to work quite well provided some sensible technique is used to select elements from G. In the author's implementation, the elements chosen are $g_1, g_2, g_3, \ldots$, where S_0 is the original set S, $a_0 = g_0$ is a random element from S_0, and inductively a_i is a random element of $S_0 - \{a_{i-1}^{-1}\}$ and $g_i = g_{i-1}a_i$; note that only one multiplication of permutations is needed to obtain g_i from g_{i-1}.

The step by step description of the random Schreier method which follows incorporates two additional features. Sometimes an upper bound b for $|G|$ is known; in this case, the algorithm terminates immediately if $\Pi_{i=1}^{k} n_i$ reaches b (in which case, $|G| = b$ and $\mathcal{B}$ and S are a base and strong generating set for G). Step 6 attempts to obtain generators of low order, involutions if possible. This step is optional. For an involutory generator, the inverse need not be stored; the same applies to a generator s of order j provided one computes $\beta^{s^{-1}}$ by applying s to β $j-1$ times.

ALGORITHM 7.1—RANDOM-SCHREIER. The notation of Definition 5.1 applies. In addition, b is an upper bound for $|G|$ ($n!$ may always be used) and N is a positive integer. The algorithm extends $\mathcal{B}$ and S; each additional element increases $|U^{(k)}U^{(k-1)} \cdots U^{(1)}|$.

1. Delete the identity from S if present.
   Set $m_0 = m$.
   For $l = 1, 2, \ldots, m_0$:
   If s_l fixes $\beta_1, \beta_2, \ldots, \beta_k$, set $k = k + 1$, $\beta_k =$ any point of Ω moved by s_l.
   If $s_l^{-1} \notin S$, set $m = m + 1$, $s_m = s_l^{-1}$.
   (This assures that Hypothesis 5.2 holds.)
2. For $i = 1, 2, \ldots, k$, apply Algorithm 4.2—ORBIT($\Omega, S^{(i)}, \{\beta_i\}, n_0, \rho, n_i, \delta, v^{(i)}$).
   If $\Pi_{l=1}^{k} n_l = b$, algorithm terminates.
   (This computes n_i and constructs $v^{(i)}$.)
3. Set $n = 0$.
   (n will denote the number of consecutive times that the condition $g \in U^{(k)}U^{(k-1)} \cdots U^{(1)}$ has held.)
4. Choose any element g of G.
   Apply Algorithm 5.6—ELEMENT($\Omega, S, \mathcal{B}, \{v^{(l)}\}, 1, g, j, \{u^{(l)}\}, h$).
   (This writes $g = h\bar{u}_{j-1}\bar{u}_{j-2} \cdots \bar{u}_1$ with $1 \leqslant j \leqslant k+1$, $\beta_l^h = \beta_l$ for $l \leqslant j-1$, $u^{(l)} \in U^{(l)}$ for all l, and either
   (1) $j \leqslant k$, $\beta_j^h \notin \Delta^{(j)}$,
   (2) $j = k+1$, $h \neq 1$, or
   (3) $j = k+1$, $h = 1$.)
5. If (3) holds in step 4:
   (a) Set $n = n + 1$.
   (b) If $n < N$, go to step 4; otherwise algorithm terminates.
6. (Optional)
   (a) If (1) holds in step 4, set $d =$ largest divisor of $|h|$ for which $\beta_j^{h^d} \notin \Delta^{(j)}$.

If (2) holds, set d = largest proper divisor of $|h|$.

(b) Set $h = h^d$.

7. Set $m = m + 1$, $s_m = h$.
   If $h^2 \neq 1$, set $m = m + 1$, $s_m = h^{-1}$.
   If (2) holds in step 4, set $k = k + 1$, β_k = any point of Ω moved by h.
   For $l = 2, 3, \ldots, j$, apply Algorithm 4.2–ORBIT($\Omega, S^{(l)}, \{\beta_l\}, n_0, \rho, n_l, \delta, v^{(l)}$).
   (This extends S and, if necessary, $\mathcal{B}$ and reconstructs $v^{(l)}$ and n_l when necessary.)
8. If $\Pi_{l=1}^{k} n_l = b$, algorithm terminates; otherwise go to step 4.

Sometimes, when using the random Schreier method, we know in advance (i) a base $\mathcal{B}'$ for G and possibly even (ii) a strong generating set S' for G relative to the base $\mathcal{B}'$. In case (i), steps 4 and 5 may be performed by applying words to $\mathcal{B} \cup \mathcal{B}'$; only when an element is added to S (step 7) must entire permutations be multiplied. In case (ii), the element g in step 4 may be chosen at random from G, using the method described in Section 5; also b may be set to $|G|$ (computable by Algorithm 4.2).

Case (i) arises in determining the subgroup G generated by a given subset S of a group G' with known base and strong generating set $\mathcal{B}'$ and S'; in this situation, of course, $\mathcal{B}'$ is also a base for G. Case (ii) arises in changing the base for G. Sims gives one method for changing the base in [6]; the random Schreier method provides an alternate approach.

The author has not conducted systematic tests of the random Schreier method; however, it appears to work extremely well on a wide variety of groups, solvable as well as nonsolvable. Sometimes the value of N must be larger than would be required, if truly random group elements were available. In the author's experience, a value of 15 to 20 has usually been sufficient, with twice that number necessary in a few cases.

8. Implementation of Interruptible Coset Enumeration. This section describes the author's implementation of an interruptible coset enumeration program. As this program is a variation of the (noninterruptible) Hasselgrove-Leech-Trotter approach described in [1], only features differing from [1], including those related to interruption, are treated in detail.

In the HLT method, a doubly linked list of cosets is maintained, and new cosets are added at the end of the list. The algorithm operates in one of two modes–the define mode or the lookahead mode; initially it is in the define mode. In the define mode, cosets are processed in the order in which they are introduced. Processing a coset consists of tracing each relator from the coset. During the tracing, new cosets are introduced, as necessary, to complete the cycle. If no more cosets can be defined, because the table is full, the algorithm switches to the lookahead mode. In the lookahead mode, no new cosets are introduced, but deductions and collapses are handled as in the define mode. The algorithm remains in lookahead mode until the first collapse occurs (for incremental lookahead) or until the end of the coset table is reached (for complete lookahead). If no collapse occurs during the lookahead pass, the algorithm terminates "unsuccessfully"; otherwise it reenters the define mode. The algorithm terminates "successfully" when every relator has been traced from every coset. Then

the coset table is closed and (c.3) is impossible (see Lemma 3.2), provided that every generator appears in some relator. (Relators of the form gg^{-1} may be added, as necessary, to insure this condition.)

The use of lookahead turns out to play a critical role in the performance of the Schreier-Todd-Coxeter-Sims algorithm; however, neither complete nor incremental lookahead yields particularly good results. In the author's coset enumeration program, lookahead is controlled by three parameters l_1, l_2, and l_3, where $l_1 \geqslant 0$ and $l_2 \geqslant l_3 \geqslant 1$. In (l_1, l_2, l_3) lookahead:

(a) The algorithm enters the lookahead mode when a relator cannot be traced because no more cosets may be defined.

(b) The algorithm remains in the lookahead mode until one of the following occurs:

(i) l_1 cosets have been processed during the lookahead pass.

(ii) l_2 or more cosets have been eliminated through collapse during the lookahead pass.

(c) Upon leaving the lookahead mode, the algorithm reenters the define mode if l_3 or more cosets were eliminated during the lookahead pass and terminates otherwise.

Note that no lookahead, complete lookahead, and incremental lookahead correspond, in the author's program, to (0, 1, 1), (∞, ∞, 1), and (∞, 1, 1) lookahead, respectively. A value of l_2 between 1 and ∞ yields a cross between incremental and complete lookahead. Both incremental and complete lookahead will be referred to as unlimited lookahead, because there is no limit (other than the size of the coset table) on the number of cosets that may be processed during a lookahead pass. Such a limit may be imposed by choosing a finite value for l_1 (limited lookahead). Normally l_3 is 1; however, a higher value may be used to terminate the enumeration in the event that a long lookahead pass results in elimination of only a small number of cosets (in which case, it may not be worthwhile to continue).

The author's coset enumeration program (mainly in Fortran) is controlled by several input parameters, including a coset table (with doubly linked list), an array REL containing the subgroup generators followed by the relators, a variable RSTART specifying the index in REL at which the relators begin, a variable MAXCOS giving the maximum size of the coset table, and variables COS, POS, and LSTTRC. Cosets are numbered 1, 2, . . . , MAXCOS, with 1 and 0 corresponding to ι and o of Section 3 respectively; coset 1 always comes first on the linked list.

Tracing of relators begins with coset COS and with the relator beginning at REL(POS). During the tracing process, COS and POS are constantly updated to point to the current coset and relator; when COS is incremented to the next coset on the linked list, POS is reset to RSTART. Tracing of relators continues until one of the following occurs:

(1) COS reaches the end of the linked list of cosets.

(2) Fewer than l_3 cosets are eliminated during a lookahead pass, as in (c) above.

(3) All relators have been traced from each coset up to and including LSTTRC (applicable only if LSTTRC $\neq$ 0).

Condition (3) is useful in resuming enumerations; should coset LSTTRC be eliminated during collapse, LSTTRC is reset to the previous coset on the linked list.

A new enumeration is initiated by calling the coset program with a trivial coset table and with COS and POS set to 1 and LSTTRC set to 0. Simple resumption of an enumeration is effected by calling the coset program with COS and POS retaining their values at the time of interruption. Resumption with added relators or subgroup generators poses additional problems. Perhaps the simplest method is to invoke the coset program twice. In the first call, REL is set to contain the new subgroup generators or relators, COS and POS are set to 1, and LSTTRC is set to the predecessor in the linked list of the value of COS at the time of interruption. In the second call, REL is set to contain all subgroup generators and relators, COS retains the value it had when the first call terminated, and LSTTRC is set to 0; the value of POS depends on how the new subgroup generators or relators are inserted into REL (POS may always be set to 1).

9. Implementation of the Schreier-Todd-Coxeter-Sims Algorithm. The author's implementation of the Schreier-Todd-Coxeter-Sims algorithm follows the step-by-step description of Algorithm 6.2 and the subalgorithms which it calls. One exception involves step 5.6(b), which is combined with step 5.2; that is, STCS does not call EQUIV directly, but immediately upon resumption of the enumeration, the new relator is traced from coset 1, resulting in the collapse required in step 5.6(b). Algorithm 6.2 allows the use of any interruptible Todd-Coxeter algorithm in step 5.2, the selection of any function M with $M(x) > x$ for all x, and the choice of any new base point β_k in steps 1 and 6. The author's program accepts input parameters $a_1, a_2, l_1, l_2, l_3, k, k'$, and $\{\beta_j\}_{j=1}^{k'}$. Step 5.2 invokes an interruptible HLT coset enumeration algorithm with (l_1, l_2, l_3) lookahead, as described in Section 8. In a resumed enumeration, the algorithm is actually invoked twice, as mentioned near the end of Section 8. The function $M(x)$ is taken to be $a_1x + a_2$. (If $a_1n_i + a_2 < n_i + l_3$, then $M(n_i)$ is increased to $n_i + l_3$ in order to insure that $|\Lambda^{\#}| > n_i$ in step 5.2, should the enumeration terminate without a closed table.) The parameters k and $\{\beta_j\}_{j=1}^{k}$ specify an initial segment of the base, exactly as in Definition 5.1; k may have a value of 0. Often k' equals k; however, it is possible to specify a higher value of k' and additional points β_{k+1}, $\beta_{k+2}, \ldots, \beta_{k'}$. These additional points become the "preferred" points to add to the base, should expansion be necessary. The exact procedure for incrementing k and choosing the new base point in step 6 is given below; for step 1, replace h by s_l in the procedure.

Set $k = k + 1$.

If h moves some point of $\{\beta_k, \beta_{k+1}, \ldots, \beta_{k'}\}$, set l to the smallest integer with $k \leqslant l \leqslant k'$ and $\beta_l^h \neq \beta_l$, and set $\gamma = \beta_l$; otherwise set $l = k'$ and γ to the smallest element of Ω with $\gamma^h \neq \gamma$.

For $t = l, l-1, \ldots, k+1$, set $\beta_t = \beta_{t-1}$.

Set $\beta_k = \gamma$.

Other input parameters include the initial generating set for G, specified by m and S, and the initial set R of relators (usually empty). R is given in the form of an

array REL containing the relators, each relator preceded by its length; a relator of length zero terminates the list. The program rearranges the array REL, if necessary, so that relators in $R^{(i)}$ precede those in $R - R^{(i)}$ ($i = 1, 2, \ldots, k$). When new relators are added to R, they are inserted into the array REL so that this condition continues to hold. An array RELEND is maintained such that RELEND(i) is the end of the last relator in $R^{(i)}$.

Three input parameters are used to impose bounds on certain quantities in the algorithm. These parameters are used in dimensioning a number of arrays. MAXBAS, MAXGEN, and MAXREL give maximum values for the base size k, the number m of strong generators, and the total length of the relator vector REL, respectively. If the algorithm cannot proceed without exceeding one of these bounds, it terminates with a return code set to indicate the cause. The bound MAXREL is particularly useful in eliminating wasted computer time in situations in which there is little hope of completion with available resources, as will be discussed in the next section.

10. Performance of the Schreier-Todd-Coxeter-Sims Algorithm. This section is devoted to evaluating the performance of the Schreier-Todd-Coxeter-Sims algorithm. Two factors definitely of interest are memory usage and execution time. If STCS is invoked for the purpose of obtaining a presentation, the nature of the presentation obtained also may be of interest. To the author's knowledge, STCS is the only currently available, general purpose algorithm for finding a base and strong generating set which is effective for permutation groups with small bases and degrees in the 100 to 10000 range. (A number of special techniques exist, but they depend heavily on the structure of the particular group.) The purpose of this section is to explore the capabilities and limitations of the STCS algorithm and to investigate how its performance varies with selection of the input parameters and other changes in strategy, with a view toward obtaining optimal performance from the algorithm.

A. *Memory Usage.* Memory requirements of the STCS algorithm are easy to approximate; for transitive groups of high degree, they are determined primarily by the arrays whose dimensions involve the degree n. In this section it will be assumed that the groups being considered are transitive on Ω. Also, it will be assumed that coset enumeration is performed with a doubly linked list, given by $\mathfrak{f}$ and $\mathfrak{b}$ as in Section 3; then the major arrays required in an enumeration are $\mathfrak{f}$, $\mathfrak{b}$, and f. For Algorithm 6.2, the arrays whose dimensions involve n (at least when $i = 1$) are given below.

Array	Dimension(s)
S	$n \times m$
$\{\nu^{(l)}\}_{l=1}^{k}$	$n \times k$
f	$M(n) \times m$
$\mathfrak{f}$	$M(n)$
$\mathfrak{b}$	$M(n)$
θ	n
δ	n
ν	$M(n)$

Note that the values of k and m above must be the final (i.e. largest) values of these variables; depending on the method of storage allocation, it may be necessary to allocate memory based on the assumption that k and m will attain the values MAXBAS and MAXGEN, respectively. A temporary array of dimension n is required in a few places, but this can overlap some of the arrays above. Thus the major arrays for STCS involve $(m + k + 2)n + (m + 3)M(n)$ entries. By contrast, the Schreier-Sims and random Schreier methods use only the arrays S and $\{v^{(l)}\}_{l=1}^{k}$ plus one or two temporary arrays of dimension n–a total of perhaps $(m + k + 1)n$ entries.

The relatively heavy memory requirements of the STCS algorithm may be minimized by several methods. One method is to choose a function M with $M(x)$ only slightly greater than x. Fortunately, this can be done without increasing execution time, provided that sufficient lookahead is used; this will be discussed in Section 10B.

A second method is to minimize the size m of the strong generating set which is produced. To some extent, a judicious choice of the initial segment to the base can help to minimize m; if one or more of the original generators fix the first point or two of the base, fewer additional generators may be needed to obtain strong generation. However, the primary problem is that the STCS algorithm (as well as the Schreier-Sims and random Schreier algorithms) tends to produce strong generating sets with redundant elements. Each generator added is nonredundant (for strong generation) when added, but it may become redundant as further generators are added. One solution is first to use the random Schreier method (with its relatively low memory requirements) to produce a probable base and strong generating set, then to employ the method described in [6] to remove redundant strong generators, and finally to apply STCS to verify the base and strong generating set. In this way, the final value of m for STCS is minimized; moreover, it is known in advance, assuming that the random Schreier method really produces a valid base and strong generating set. It turns out, fortunately, that this approach also yields the best execution times.

A third method for minimizing memory usage is to keep only certain of the major arrays in memory at one time; the remaining arrays can reside on an external storage device. For example, only one of the two largest arrays–S and f–might be held in memory at once. The problem with this approach is that step 5.4.3 accesses f and S, alternately, a large number of times, resulting in an excessive number of input/output operations. If, however, the entire Schreier vector v for the coset table T is constructed first in step 5.4, then all references to f can be placed before any reference to S. The changes to step 5 of Algorithm 6.2 are as follows:

(a) Before step 5.1, write out S and $\{v^{(l)}\}_{l=1}^{k}$ on external storage.

(b) Before returning from step 5.3 to step 5(a), read in S and $\{v^{(l)}\}_{l=1}^{k}$ from external storage.

(c) Replace steps 5.4.1 through 5.4.5 by the following.

5.4.1′. Write out f_0 and b on external storage.

5.4.2′. Apply Algorithm 4.2–ORBIT$(\Lambda, \hat{S}^{(i)}, \{1\}, n_0, \rho, l, \delta, v)$–to construct δ and a Schreier vector v for T (Algorithm 4.2 may be terminated as soon as $\delta_{n_{i+1}}$ has been set in step 5).

5.4.3′. For $t = 2, 3, \ldots, n_i + 1$, set $v(\delta_t) = f(\delta_t, v(\delta_t)')$.

5.4.4′. Write out f on external storage; read in S from external storage.

5.4.5′. Set $t_0 = 1$, $\theta(\beta_i) = 1$, $\theta(\gamma) = 0$ for $\gamma \in \Delta^{(i)} - \{\beta_i\}$, $\theta'(1) = \beta_i$.

5.4.6′. Set $t_0 = t_0 + 1$, $\tilde{\gamma} = \delta_{t_0}$, $\tilde{\alpha} = v(\tilde{\gamma})$, $\alpha = \theta'(\tilde{\alpha})$, $\gamma = \alpha^s$, where $s = v(\tilde{\gamma})$.

5.4.7′. If $\theta(\gamma) = 0$, set $\theta(\gamma) = \tilde{\gamma}$ and $\theta'(\tilde{\gamma}) = \gamma$ and go to step 5.4.6′.

5.4.8′. Same as step 5.4.5, but replace $f(\rho, v(\rho)')$ by $v(\rho)$ in step 4 of Algorithm 4.1.

(d) Before step 5.5, read in $\{v^{(l)}\}_{l=1}^{k}$ from external storage.

(e) Before step 5.6(b), write out S and $\{v^{(l)}\}_{l=1}^{k}$ on external storage, and read in f, $\mathfrak{f}$, and b from external storage.

Note that this modification of the STCS algorithm employs two additional arrays—θ', which is an inverse to θ, and v, which permits use of Algorithm 4.1 without access to S. It is not really necessary to construct the entire Schreier vector v in step 5.4.2′; the first $n_i + 1$ entries will always suffice. At any time during the algorithm, only arrays in one of the four groups below need reside in memory (τ denotes a temporary array of dimension n).

Arrays	Number of entries (when $i = 1$)	When in Memory
f, $\mathfrak{f}$, b	$(m + 2)M(n)$	5.1–5.4.1′, 5.6
f, v, v	$(m + 2)M(n)$	5.4.2′–5.4.4′
S, v, v, δ, θ, θ'	$(m + 2)n + 3M(n)$	5.4.5′–5.4.8′
S, $\{v^{(l)}\}_{l=1}^{k}$, τ	$(m + k + 1)n$	everywhere else

Thus memory requirements for the major arrays are reduced from $(m + k + 2)n + (m + 3)M(n)$ to $\text{MAX}((m + 2)M(n), (m + 2)n + 3M(n), (m + k + 1)n)$; a level comparable to those for the Schreier-Sims and random Schreier methods. If, for example, $M(n) = 1.2n$, $k = 5$, and $m = 10$, these requirements are reduced from $33n$ to $16n$, approximately a 50% reduction.

The disadvantage of this modification of Algorithm 6.2 is that $n_i + 1$ entries of v and v must be constructed in steps 5.4.2′ and 5.4.3′, even though only a small fraction of these entries may later be used. Perhaps, when n_i is large, one should construct only a fraction of v and v, say $c(n_i + 1)$ entries where $c < 1$. Should t_0 exceed $c(n_i + 1)$ in step 5.4.6′, it would be necessary to write out S and read in f, extend v and v, and write out f and read in S before proceeding with step 5.4.6′. As a guide to the choice of c, statistics on the fraction of the Schreier vector actually used for various groups are given in Section 10B.

Performance of this modification of the STCS algorithm, together with other variations of the algorithm, will be studied in a forthcoming article by the author.

B. *Execution Time and Length of Defining Relators.* This section investigates performance of the STCS algorithm with respect to execution time and length of the defining relators. If STCS is used in order to obtain a presentation, one might hope

for a fairly small number of reasonably short relators; hence, the total length of all defining relators was chosen as a measure of the "desirability" of a presentation. Actual presentations obtained by the STCS algorithm for several interesting groups are given in Table 4.

The STCS algorithm was tested (a) in building up a base and strong generating set and (b) in verifying a base and strong generating set, produced by the random Schreier method. As previous experience and preliminary testing indicated that (b) gave vastly superior performance, it was investigated more extensively than (a).

Four primitive groups were chosen for the investigation. Each group was constructed by coset enumeration, using the presentations in [2], [4], and [9]. The random Schreier method was employed to build up a base and strong generating set; the first several base points were chosen at random. The procedure in [6] was used to delete redundant strong generators (including some of the original generators). The results were strong generating sets having little resemblance to the original generating sets. (Conceivably the algorithm might work unusually well with the original sets because of the short presentations involving them.) The specific groups used were as follows:

Group	Order	n	Point stabilizer	m	k	Basic orbit lengths
Hall-Janko	604,800	315	centralizer of involution	8	4	315, 80, 6, 4
Held	4,030,387,200	2058	$Sp_4(4) \cdot Z_2$	6	4	2058, 1360, 144, 10
M_{24}	244,823,040	2024	stabilizer of 3-element set	5	3	2024, 1120, 108
Higman-Janko-McKay	50,232,960	6156	$SL_2(16) \cdot Z_2$	6	4	6156, 85, 32, 3

These permutation groups were used to test STCS in verifying a base and strong generating set. In addition, from each group, two-element subsets were selected at random, using the method described in Section 5, until five two-element generating sets were obtained. To test STCS in building up a base and strong generating set, a null initial base segment ($k = 0$) and these random 2-element generating sets were extended to bases and strong generating sets; in each case k' was set to n and $\beta_1, \beta_2, \ldots, \beta_{k'}$ to a random permutation of $1, 2, \ldots, n$.

First we study performance of STCS in verifying strong generation; in particular, the influence of the function $M(x) = a_1 x + a_2$ and the type (l_1, l_2, l_3) of lookahead will be investigated. Table 1 gives results for a wide variety of values of a_1 and l_1, both with $a_2 = 0$ and $a_2 = 50$ ($l_2 = l_3 = 0$ in all cases). It is immediately evident that a_1 and l_1 have a great effect on performance; l_1 is especially important. Note the tremendous differences in performance between $l_1 = 0$ and $l_1 = 10$ whenever $a_1 \leqslant 1.5$. Consider the upper half or lower half of the table for one group (a_2 fixed). The primary variation in performance occurs in moving from upper left to lower right in the half table. For each half table, there is a region near the center, and extending toward the lower left and upper right, where performance is near optimal with respect

to both execution time and total relator length. Moving down or right from this region (i.e. increasing a_1 or l_1) causes execution time to increase steadily, while total relator length either remains constant or decreases slightly. Moving up or left from this region (i.e. decreasing a_1 or l_1) causes both execution time and total relator length to increase drastically; sometimes the algorithm fails to complete even with MAXREL = 4000. (Higher values of MAXREL lead to exceedingly long execution times.) Several conclusions may be drawn from Table 1.

TABLE 1

Performance of the STCS algorithm with $M(x) = a_1x + a_2$ *and* $(l_1, 1, 1)$ *lookahead (verifying strong generation)*

Table entries have the form t (L), where
t = execution time in seconds (IBM 370/158),
L = total length of the defining relators.
An asterisk indicates STCS failed to complete with MAXREL = 4000.

HALL-JANKO GROUP (degree 315)

a_1	a_2 \ l_1	0	10	25	50	100	∞
1.00	1	27.6 (2238)	3.4 (304)	2.2 (158)	2.4 (98)	2.1 (100)	3.3 (100)
1.10	0	20.3 (2243)	2.2 (120)	2.0 (107)	2.7 (107)	2.6 (106)	3.4 (106)
1.20	0	11.5 (1714)	2.5 (182)	1.9 (96)	2.6 (96)	2.4 (96)	3.7 (96)
1.33	0	10.6 (1530)	2.0 (98)	1.9 (98)	2.6 (100)	2.6 (100)	4.1 (100)
1.50	0	5.2 (652)	2.0 (98)	2.0 (100)	2.7 (100)	2.4 (100)	4.1 (100)
2.00	0	2.2 (98)	2.2 (108)	2.2 (100)	2.8 (100)	2.9 (100)	5.1 (100)
3.00	0	2.5 (100)	2.5 (100)	2.7 (100)	3.8 (100)	3.3 (100)	6.5 (100)
5.00	0	3.4 (100)	3.6 (100)	3.7 (100)	4.8 (100)	4.8 (100)	13.8 (100)
1.00	50	16.8 (2210)	2.8 (232)	1.9 (98)	2.5 (100)	2.4 (100)	3.6 (100)
1.10	50	11.9 (1729)	2.6 (186)	2.0 (98)	2.8 (100)	2.6 (100)	4.6 (100)
1.20	50	10.1 (1497)	2.1 (98)	2.1 (100)	2.8 (100)	2.6 (100)	4.3 (100)
1.33	50	4.9 (633)	2.0 (98)	2.1 (98)	2.7 (100)	2.7 (100)	4.7 (100)
1.50	50	4.0 (432)	2.1 (98)	2.1 (100)	2.7 (100)	2.7 (100)	4.5 (100)
2.00	50	2.2 (98)	2.3 (100)	2.5 (100)	3.0 (100)	3.2 (100)	5.7 (100)
3.00	50	2.5 (100)	2.7 (100)	2.7 (100)	3.5 (100)	3.4 (100)	8.6 (100)
5.00	50	3.5 (100)	4.0 (100)	3.9 (100)	5.0 (100)	5.4 (100)	15.8 (100)

HELD'S GROUP (degree 2058)

a_1	a_2 \ l_1	0	10	25	50	100	∞
1.00	1	*	89 (2241)	51 (1013)	50 (1071)	23 (294)	47 (294)
1.10	0	*	35 (907)	31 (643)	19 (294)	21 (294)	59 (294)
1.20	0	*	46 (1158)	20 (294)	20 (294)	21 (294)	75 (294)
1.33	0	*	27 (555)	21 (315)	21 (315)	25 (315)	95 (315)
1.50	0	*	20 (315)	21 (307)	21 (315)	26 (315)	93 (299)
2.00	0	114 (3240)	21 (315)	22 (315)	23 (315)	30 (315)	118 (299)
3.00	0	29 (398)	23 (315)	28 (315)	32 (315)	47 (299)	
5.00	0	30 (315)	35 (315)	40 (315)	42 (299)	64 (315)	
1.00	50	*	54 (1457)	24 (433)	22 (347)	22 (294)	
1.10	50	*	38 (1026)	18 (294)	19 (294)	22 (294)	
1.20	50	*	33 (788)	19 (307)	19 (294)	23 (294)	
1.33	50	*	30 (712)	20 (306)	20 (315)	27 (315)	
1.50	50	*	30 (712)	22 (315)	21 (315)	26 (315)	
2.00	50	111 (3250)	19 (307)	23 (315)	28 (315)	32 (315)	
3.00	50	24 (339)	23 (315)	37 (315)	39 (315)	49 (299)	
5.00	50	29 (315)	34 (315)	42 (315)	50 (299)	63 (315)	

TABLE 1 (*continued*)

M_{24} (degree 2024)

a_1	a_2 \ l_1	0	10	25	50	100	∞
1.00	1	*	52 (1367)	37 (880)	13 (238)	16 (259)	21 (177)
1.10	0	*	19 (428)	14 (280)	11 (177)	12 (177)	20 (177)
1.20	0	*	27 (752)	14 (301)	12 (217)	12 (177)	22 (177)
1.33	0	*	18 (450)	12 (239)	12 (217)	13 (177)	23 (177)
1.50	0	90 (3240)	19 (480)	11 (217)	12 (217)	12 (177)	36 (165)
2.00	0	22 (637)	17 (375)	12 (217)	12 (217)	12 (177)	37 (165)
3.00	0	13 (217)	13 (217)	13 (225)	14 (217)	14 (177)	46 (165)
5.00	0	15 (185)	15 (185)	16 (185)	16 (217)	18 (185)	79 (165)
1.00	50	*	30 (790)	18 (441)	12 (217)	14 (200)	
1.10	50	*	22 (544)	13 (238)	12 (217)	14 (177)	
1.20	50	*	16 (385)	12 (217)	12 (217)	12 (177)	
1.33	50	*	16 (384)	12 (217)	12 (217)	12 (177)	
1.50	50	92 (3011)	16 (388)	12 (217)	12 (217)	12 (177)	
2.00	50	24 (637)	16 (375)	12 (217)	13 (217)	13 (217)	
3.00	50	13 (217)	13 (217)	13 (225)	14 (225)	14 (177)	
5.00	50	16 (185)	16 (185)	15 (185)	16 (185)	19 (185)	

HIGMAN-JANKO-MCKAY GROUP (degree 6156)

a_1	a_2 \ l_1	0	10	25	50	100	250	∞
1.00	1	*	*	132 (1213)	57 (407)	48 (296)	46 (202)	95 (296)
1.00	50	*	299 (3007)	68 (566)	39 (227)	45 (238)	44 (180)	88 (177)
1.20	50	*	225 (2659)	61 (514)	49 (357)	40 (201)	42 (175)	109 (177)
2.00	50	*	69 (535)	41 (178)	42 (178)	49 (178)	70 (178)	
3.00	50	46 (178)	46 (178)	47 (178)	51 (178)	57 (178)		

(i) For each of the four groups, optimal performance can be obtained with small values of $M(x)$, say $1.1x$ or $1.2x$, provided that adequate lookahead is employed (i.e. l_1 sufficiently large). Reasonably good performance can be obtained even when $M(x)$ has its smallest possible value, namely $x + 1$. This is important, since small values of $M(x)$ minimize memory usage.

(ii) Without lookahead (i.e. $l_1 = 0$), the algorithm is nearly useless, unless memory size is adequate to permit large values of a_1 (Held's group and the Higman-Janko-McKay group require a_1 about 3 for reasonably good performance).

(iii) Unlimited lookahead (i.e. $l_1 = \infty$) yields poor execution times. If it is to be employed, a_1 should be chosen very close to 1.

(iv) With $a_1 = 1.1$ or 1.2, the value of l_1 should increase as the degree n increases; however, l_1 should be less than linear in n. For the four groups studied, at least, $l_1 = c\sqrt{n}$, with c between 1 and 2, yields good performance.

(v) Increasing a_1 or l_1 beyond the values giving optimal execution times seems to produce little if any reduction in total relator length.

(vi) Unless $a_1 = 0$, the influence of a_2 on performance appears to be minimal.

Table 2 gives results for a wide variety of values of (l_1, l_2, l_3), with $M(x)$ fixed at $1.2x + 50$. The objective here is to investigate the effect of choosing $l_3 > 1$. From

the table, it appears that, when l_1 is chosen too large, higher values of l_3 reduce execution time significantly but not to optimal levels. However, when l_1 is chosen correctly, higher values of l_3 produce no improvement and sometimes even increase execution time. On balance, there appears to be little advantage to values of l_3 greater than 1.

TABLE 2

Performance of the STCS algorithm with $M(x) = 1.2x + 50$ *and* (l_1, l_2, l_3) *lookahead (verifying strong generation)*

Table entries are as in Table 1.

l_1	l_2	l_3	HALL-JANKO GROUP (degree 315)	HELD'S GROUP (degree 2058)	M_{24} (degree 2024)	HIGMAN-JANKO-MCKAY GROUP (degree 6156)
0	1	1	10.1 (1497)	*	*	*
10	1	1	2.1 (98)	33 (788)	16 (385)	225 (2659)
10	2	2	2.3 (98)	32 (774)	17 (385)	163 (1743)
25	1	1	2.1 (100)	19 (307)	12 (217)	61 (514)
25	3	3	2.3 (100)	24 (491)	12 (217)	72 (643)
25	5	5	2.4 (100)	20 (334)	12 (217)	100 (952)
50	1	1	2.8 (100)	19 (294)	12 (217)	49 (357)
50	5	5	2.7 (100)	19 (294)	12 (217)	72 (596)
50	10	10	2.5 (100)	20 (307)	12 (217)	58 (461)
100	1	1	2.6 (100)	23 (294)	12 (177)	40 (201)
100	10	10	2.9 (100)	20 (294)	12 (217)	78 (568)
100	20	20	2.5 (100)	20 (294)	12 (217)	45 (263)
250	1	1	3.9 (100)	33 (294)	14 (177)	42 (175)
250	25	25	3.0 (100)	22 (294)	12 (177)	41 (175)
250	50	50	2.8 (100)	21 (294)	12 (177)	42 (210)
∞	1	1	4.3 (100)	71 (294)	28 (185)	109 (177)
∞	25	25	3.2 (100)	54 (294)	24 (185)	90 (175)
∞	50	50	3.2 (100)	46 (294)	23 (185)	79 (177)

Table 3 gives results on using STCS to build up a base and strong generating set. It is evident that performance is not nearly as good when STCS is used in this way.[10] For M_{24} there is roughly an eight-fold increase in execution time. There appear to be two reasons for this reduction in performance. The values of m are larger due to redundant strong generators and to fewer involutory generators, resulting in greater total relator lengths and correspondingly longer enumeration times. When a new generator is added and i is reset to j in steps 6 and 7 of Algorithm 6.2, the time previously spent in checking that $H_{\beta_l}^{(l)} = H^{(l+1)}$ for $l \leqslant j$ becomes wasted as these checks must be repeated.

In the course of collecting the data in Tables 1, 2, and 3, a number of more detailed statistics were obtained. Some of these yield insight into the influence of a_1 and l_1 on performance and, in particular, into the poor results obtained with too

[10] In making the comparison, the time required by the random Schreier method and by the removal of redundant strong generators should be considered; however, these times are minimal.

TABLE 3

Performance of the STCS algorithm in building up a base and strong generating set

In all cases, k = 0 initially.

For each group, data is given for five (initial) two-element generating sets selected at random from the set of two-element generating sets.

Execution times are in seconds on an IBM 370/158.

Always M(x) = 1.2x + 50, $\ell_2 = \ell_3 = 1$, and MAXREL = 4000.

Group	ℓ_1	Execution time	Total length of relators	Basic orbit lengths	m (final value)
HALL-JANKO	25	5.0	191	315, 10, 64, 3	11
		3.6	113	315, 160, 4, 3	12
		4.2	200	315, 32, 20, 3	12
		3.8	166	315, 80, 8, 3	12
		6.0	320	315, 80, 8, 3	14
HELD	50	*	*		
		*	*		
		104	1505	2058, 136, 900, 8, 2	17
		62	703	2058, 1360, 120, 12	16
		170	1348	2058, 1360, 120, 12	19
HELD	200	317	1741	2058, 425, 384, 2, 6	16
		115	1254	2058, 136, 300, 24, 2	18
		93	871	2058, 136, 900, 8, 2	17
		56	353	2058, 1360, 120, 12	16
		183	1352	2058, 1360, 120, 12	19
M_{24}	50	156	1666	2024, 630, 48, 2, 2	14
		85	1012	2024, 630, 96, 2	21
		128	1477	2024, 1120, 54, 2	15
		97	1751	2024, 210, 288, 2	13
		102	1340	2024, 1120, 54, 2	18
M_{24}	200	88	904	2024, 630, 48, 2, 2	14
		82	490	2024, 630, 96, 2	20
		99	992	2024, 1120, 54, 2	15
		81	996	2024, 210, 288, 2	13
		116	1033	2024, 1120, 54, 2	18

small or too large a combination of a_1 and l_1. The problems with a_1 and/or l_1 too large are illustrated by the following data for the Hall-Janko group (verifying strong generation).

a_1	1.1	5.0	1.1	5.0
l_1	25	25	∞	∞
Total time (sec.)	2.0	3.7	4.1	13.8
Nonenumeration time	0.6	0.7	0.6	0.7
Enumeration time	1.4	3.0	3.5	13.1
Tracing relators (define mode)	0.5	1.0	0.6	1.1
Tracing relators (lookahead mode)	0.5	0.4	2.3	10.3
Processing equivalences	0.4	1.6	0.6	1.7
Total number of cosets defined	1592	5343	2035	5323
Total number of cosets processed in lookahead mode	489	554	4484	21592

Here "nonenumeration" refers to all of Algorithm 6.2, except for the coset enumerations (steps 5.2 and 5.6(b)). In the leftmost column of data, a_1 and l_1 are chosen optimally; in the remaining three columns, one or both of them are too large. In these three cases, the increased execution times result entirely from greater time spent in coset enumeration. As a_1 increases, the total number of cosets defined increases, and correspondingly the time required to process coincidences (Algorithm 3.4) goes up. As l_1 increases, the time spent in tracing relators while in lookahead mode goes up; note the sharp rise in the number of cosets processed in the lookahead mode. A great deal of effort is wasted in lookahead during early enumerations, which cannot possibly complete since a set of defining relators has not yet been built up.

TABLE 4

Defining relators produced by the STCS algorithm

HALL-JANKO GROUP

(M(x) = 1.2x + 50 and (25,1,1) lookahead)

d^4,

$bd^{-2}b^{-1}d^{-2}$, b^3, $bdbd^{-1}bd$,

$(c^{-1}d)^2$, $cbd^{-1}c^{-1}d^{-1}b$, c^6, $c^{-2}bdbc^{-2}d^{-1}b^{-1}d$, $c^3b(dc)^2b^{-1}d^{-2}$,

$ada^{-1}d^{-1}$, $a^{-1}bad^{-1}c^{-1}d^{-1}c^2d^{-1}b$, a^3, $a^{-1}c^2b^{-1}c^{-1}a^{-1}cdc^{-2}d^{-1}b$,

$ac^2bd^{-1}ac^{-1}ad^{-1}c^{-3}bc^{-1}d$.

M_{24}

(M(x) = 1.2x + 50 and (100,1,1) lookahead)

c^2, d^6, $(cd^{-2}cd)^2$, $(cd)^6$, $(cd^{-1}(cd)^2)^2$

b^2, $bd^{-1}cdb(cd)^2cd^{-1}$, $(bc)^4$, $bdcbcd^{-1}bd^{-3}$, $bcbd^{-1}cbcdbd^{-1}cbcdc$,

a^2, $adad^{-1}abd^2bcbd^{-2}b$, $a(dc)^2d^{-1}cabcdbd^2bdbd^{-1}c$,

$abd^{-1}bdbadbd^2bd^{-2}bd^{-1}$, $abdad^{-1}babdcdcbdcd^{-3}$, $ad^{-1}bacdcacbdcbdbcbd^2c$.

When a_1 and/or l_1 are too small, a different problem occurs. A new relator will be added in step 5.6(a) of Algorithm 6.2; typically, however, only a few cosets will be deleted in the equivalence of step 5.6(b). When the coset enumeration is resumed in step 5.2, few new cosets can be introduced, and the enumeration leaves the define mode almost immediately. If l_1 is close to zero, the lookahead pass provides little if any help. Thus the enumeration terminates ($\Lambda^{\#} = M(n_i)$), with little progress having been made. A new relator is added, and the process repeats. As more and more long relators are added, it becomes even more difficult to make progress during an enumeration. Since each relator added requires verification, execution time for the nonenumeration portion of the algorithm can grow quite long. At this stage, enumeration times remain quite short, because the enumerations terminate so quickly. Eventually the STCS procedure may terminate unsuccessfully because the total relator length exceeds the bound MAXREL. If MAXREL is large enough, the critical collapse(s) will occur

eventually, and then the enumeration will proceed to completion; by this time, however, total relator length may be so great that an exceedingly long time is required to trace all relators in the final enumeration. For example, about 12 seconds were required to verify a base and strong generating set for M_{24} with a_1 and l_1 chosen optimally. With $a_1 = 1.5$ and $l_1 = 0$, 90 seconds were required, 49 seconds of non-enumeration time (probably much of it verifying relators) and 41 seconds of enumeration time. A total of 145 relators were added. For the vast majority of them, fewer than ten cosets were eliminated in the subsequent collapse (step 5.6(b)). Of the 41 seconds of enumeration time, 35 seconds were used in tracing relators; 33 of these 35 seconds were used in the final resumption of the enumeration.

One way to avoid the situation described above is to select a relatively small (but adequate) value of MAXREL—perhaps 500 to 1000 depending on the degree and order. If a_1 and l_1 are chosen too small for good performance, STCS will terminate unsuccessfully but rather quickly; one may then try again with a larger value of a_1 or (preferably) l_1. Another approach, suggested to the author by Sims, is to add relators selectively; that is, step 5.6(a) might be omitted when the total relator length grows long. Note that this does not effect the validity of the algorithm; in fact, it corresponds merely to use of a different coset enumeration algorithm in step 5.2. This approach would undoubtedly reduce enumeration times but might well increase time spent in verifying relators, already a substantial factor when a_1 and l_1 are too small.

While testing the STCS method, statistics were obtained on the amount of the Schreier vector v actually constructed in step 5.4. It may be necessary to construct as many as $n_i + 1$ entries of v before finding the words w_1 and w_2; however, when n_i is large and a_1 and l_1 are at least large enough to give optimal time performance, only a fraction of this number usually is needed. For the four groups investigated (verifying strong generation), the number of entries of v actually filled in was $c(n_i + 1)$, where c ranged from 0.03 to 0.19, whenever $n_i \geqslant 250$ and a_1 and l_1 were large enough. This suggests that, in implementing the modifications suggested in Section 10A, it might be best to construct initially fewer than $n_i + 1$ entries of v in step 5.4.2′ whenever n_i is large, as mentioned at the end of Section 10A.

Statistics also were obtained on the distribution of execution time for the STCS algorithm. For the four groups studied, whenever a_1 and l_1 were chosen to give optimal performance, the percentage of total execution time spent in coset enumeration ranged from 70% to 80%. Thus any effort toward efficient programming (for example, programming selected sections in assembler language) should be directed primarily toward the coset enumeration program.

It should be kept in mind that all results in Section 10B were obtained with an HLT coset enumeration algorithm with (l_1, l_2, l_3) lookahead. The HLT method is one of two major approaches to coset enumeration successfully implemented for large groups; see [1]. The other method is due to Felsch. Although the HLT method seems to give the best performance for a majority of groups, when coset enumeration is used in its usual way [1], it remains to be tested whether this applies also to the special way

in which enumeration is invoked in the STCS algorithm. The use of the Felsch method, as well as the modifications discussed in Section 10A and some other variations of the STCS algorithm, will be discussed in a forthcoming paper by the author.

Department of Mathematics
University of Illinois at Chicago Circle
Chicago, Illinois 60680

1. JOHN J. CANNON, LUCIEN A. DIMINO, GEORGE HAVAS & JANE M. WATSON, "Implementation and analysis of the Todd-Coxeter algorithm," *Math. Comp.*, v. 27, 1973, pp. 463–490.

2. JOHN J. CANNON & GEORGE HAVAS, "Defining relations for the Held-Higman-Thompson simple group," *Bull. Austral. Math. Soc.*, v. 11, 1974, pp. 43–46.

3. MARSHALL HALL, JR., *The Theory of Groups*, Macmillan, New York, 1959.

4. JOHN McKAY & DAVID W. WALES, "The multipliers of the simple groups of order 604,800 and 50,232,960," *J. Algebra*, v. 17, 1971, pp. 262–272.

5. CHARLES C. SIMS, "Determining the conjugacy classes of a permutation group," in *Computers in Algebra and Number Theory* (Proc. Sympos. Appl. Math., New York, 1970), SIAM–AMS Proc., Vol. 4, Amer. Math. Soc., Providence, R. I., 1971, pp. 191–195.

6. CHARLES C. SIMS, "Computation with permutation groups," in *Proc. Second Sympos. Symbolic and Algebraic Manipulation*, Assoc. Comput. Mach., New York, 1971.

7. CHARLES C. SIMS, "Some algorithms based on coset enumeration," Unpublished notes, 1974.

8. CHARLES C. SIMS, "Some group theoretic algorithms," in *Topics in Algebra*, Lecture Notes in Math., Vol. 697, Springer-Verlag, Berlin, 1978.

9. J. A. TODD, "Abstract definitions for the Mathieu groups," *Quart. J. Math.*, v. 21, 1970, pp. 421–424.

10. J. A. TODD & H. S. M. COXETER, "A practical method for enumerating cosets of a finite abstract group," *Proc. Edinburgh Math. Soc.*, v. 5, 1936, pp. 26–34.