DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume **11**, 1993

Combinatorial Tools for Computational Group Theory

GENE COOPERMAN AND LARRY FINKELSTEIN

ABSTRACT. A variety of elementary combinatorial techniques for permutation groups are reviewed. It is shown how to apply these techniques to yield faster and/or more space-efficient algorithms for problems including group membership, normal closure, center, base change and Cayley graphs. Emphasis is placed on randomized techniques and new data structures. The paper includes both a survey of recent algorithms with which the authors have been associated, and some new algorithms in the same spirit that have not previously appeared in print. Many of the results include both complexity bounds and pseudo-code, along with comments for faster software implementations.

Contents

1. Introduction
   1.1. Notation and Basic Concepts
2. Random Subproducts
   2.1. Random Schreier Subproducts: Elementary Group Membership
   2.2. Random Normal Subproducts: Elementary Normal Closure
   2.3. Reduction of Generators
   2.4. Random Prefixes: Normal Closure Revisited
3. Nearly Optimal Group Membership
   3.1. Nearly Optimal Group Membership
   3.1.1. Short Schreier Trees
   3.1.2. Cube Schreier Trees
   3.1.3. Local Expansion (Babai)
   3.2. Group Membership: Large and Small Base

1991 *Mathematics Subject Classification.* Primary 20B02, 20B04; Secondary 20P05.

The authors were supported in part by NSF Grants # CCR-8903952 and CCR-9204469.

This paper was published in the *Proceedings of the DIMACS Workshop on Groups and Computation*, DIMACS-AMS **11**, 1993, pp. 53–86.

3.3. Elementary Near Optimal Group Membership
4. Base Change
5. Reduction of Other Constructions to Group Membership
6. Compact Data Structures: Cayley and Schreier Coset Graphs
7. Software Implementations
A. Randomized Algorithms: Terminology (Appendix)

1. Introduction

The traditional approach in developing algorithms for computing with permutation group has been to study several disjoint problems such as *group membership*, *base change* and *normal closure* as distinct computational problems. While certain themes do appear in more than one context, such as the point stabilizer sequence, or possibly a base change, it has not been clear that progress on one front would necessarily result in progress on a different front. This had been our own viewpoint while searching for improved low-level data structures and algorithmic "subroutines". Each of those "building blocks" was tailored to a particular higher goal, such as group membership [**5, 6, 21**], base change [**18, 22**], a strong generating test [**17**], normal closure [**20**], and spanning lattices for Cayley graphs [**16**].

We no longer view these "building blocks" as algorithmic subroutines, but as solutions to fundamental problems in their own right. These fundamental problems tend to be the bottlenecks in deriving faster algorithms for the higher level problems.

Three examples suffice to illustrate the point. The first is the construction of a suitable data structure for the computation of coset representatives for G/G_x, where G is a permutation group acting on Ω and $x \in \Omega$. The choice of such a data structure is a critical component of most group membership algorithms. The tradeoff typically made in designing such a data structure is the time to recover an arbitrary coset representative versus the number of permutations which must be stored to accomplish this task. Finding the right balance is crucial in reducing the time complexity of group membership while still working within reasonable space constraints and motivated the work described in [**6**].

The second example is the construction of group elements with properties similar to random group elements [**5, 22, 24**]. It has long been known that the availability of truly random group elements leads to a very efficient group membership test [**2, 29, 22**].

The third example is reduction of a generating set to one of smaller size. Construction of a chain of subgroups often leads to combinatorial explosion in the number of generators. Although it has been known for some time how to manage this in the case of permutation groups, the methods developed in [**5**] are sufficiently general to be applied to other domains, such as matrix groups over finite fields. This is useful for many algorithms which test for a certain

property by successively constructing generators for a chain of subgroups. This happens, for instance, in testing for solvability, where one must compute the normal closure of the set of commutators formed from the generating set for each successive subgroup in the derived series. Unless care is taken, the number of generators for each subgroup in the derived series will be at least the square of the number in its predecessor.

This paper illustrates this philosophy through a survey of recent results. Some have been previously published and some are new. Those methods that previously appeared in the literature are only sketched here, with pointers to a fuller description in the literature. Greater detail is provided for the "new" results. To fix notation, The permutation group is denoted G, and its degree is n. The new results include:

- an elementary $O(n^4 \log^2 n)$ group membership algorithm with at least $1 - \exp(-n)$ reliability (§2.1).
- an elementary $O(n^3)$ normal closure algorithm (with generators as output) with at least $1 - \exp(-n)$ reliability (§2.2).
- a single, moderately elementary $O(n^2 \log |G| \log n)$ group membership algorithm with at least $1 - 1/n$ reliability (§3.3).

The new results were chosen to emphasize the intuitive simplicity and purely combinatorial nature of the tools involved. The final complexity often represents only a modest improvement or alternative development as compared to our previously published algorithms [**5, 6**]. The new results may also lend themselves to faster implementations in some situations. We distinguish between *small base groups* (families of groups satisfying $\log |G| = O(\log^c n)$ for some constant c) and *large base groups* (all other families of groups).

We frequently employ the notation $O^{\sim}(f(n)) \equiv O((f(n))\,(\log^{c'} n))$ for some constant c' (read as "soft oh" of $f(n)$). We also employ the related notation $g(n) = \Omega(f(n))$ and $g(n) = \Theta(f(n))$. $g(n) = \Omega(f(n))$ if and only if $1/g(n) = O(1/f(n))$ and $g(n) = \Theta(f(n))$ if and only if both $g(n) = \Omega(f(n))$ and $g(n) = O(f(n))$. The reliabilities have been included to better estimate the time-reliability tradeoffs. More detailed asymptotic estimates are provided in the body of the paper.

§2 develops the theme of random subproducts, a solution to the second of the three "building block" examples above. As an immediate application, we describe the elementary group membership and normal closure algorithms referred to above. These continue to be the fastest algorithms with exponential reliability, although Babai, Luks and Seress have recently announced a deterministic group membership algorithm with running time $O^{\sim}(|S|n^3)$.

§3 reviews our recent work on group membership algorithms for both large [**5**] and small [**6**] base, which are nearly optimal in a complexity sense made precise in that section. This work is joint with Babai, Luks and Seress. Additional combinatorial techniques, which were required to achieve those results, are developed first.

§3 also describes a near optimal elementary group membership algorithm with further small improvements in complexity. The new algorithm is particularly interesting because it continues a trend towards simplifying the mathematical basis for group membership algorithms. Babai, Luks and Seress had introduced a $O^{\sim}(n^4)$ algorithm [**8**] by using the classification of finite simple groups, and a structure forest analysis in terms of transitivity and primitivity. The $O(n^3 \log^4 n)$ Monte Carlo algorithm introduced in [**5**] requires only the structure analysis. The new approach leads to a $O(n^3 \log^2 n)$ Monte Carlo algorithm and requires neither the classification nor the structure analysis. By introducing a variant of Sims's "sifting" (or "stripping"), the new algorithm returns to some of the simplicity of Sims's original $O(n^6)$ algorithm [**35**].

Next, §4 reviews a sequence of results on base changes which are especially useful for the case of small base groups. §5 reviews joint work with Luks allowing transformations of other problems into group membership, often via a base change. §6 discusses a non-polynomial-time algorithm for computing and storing Cayley graphs and Schreier coset graphs, which we have found important in applications. The more applied tone is continued in §7 with a short discussion of our experience at software implementation of some algorithms presented in this paper.

1.1. Notation and Basic Concepts.

Let G be a permutation group acting on an n-element set Ω with G specified by a generating set S, and let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be a fixed ordering of the points of Ω. The *point stabilizer sequence* of G relative to α is the chain of subgroups

$$G = G^{(1)} \supseteq G^{(2)} \cdots \supseteq G^{(n)} = \{1\}$$

where $G^{(i)} = G_{\alpha_1,\ldots,\alpha_{i-1}}$, $1 \leq i \leq n$. S is called a *strong generating set* for G relative to α if

$$\langle S \cap G^{(i)} \rangle = G^{(i)},\ 1 \leq i \leq n.$$

The central problem for group membership is the construction of a strong generating set. This allows one to efficiently construct a right transversal $T^{(i)}$ for $G^{(i)}/G^{(i+1)}$ for each i, a set of right coset representatives for $G^{(i)}/G^{(i+1)}$. Construction of a strong generating set S is usually carried out through a variation of Schreier generators. Given a transversal $T^{(i)}$ and generators $S^{(i)}$ for $G^{(i)}$, $G^{(i+1)}$ is generated by the *Schreier generators*

$$\{tg\overline{tg}^{-1}\colon\ t \in T^{(i)}, g \in S^{(i)}\},$$

where $\overline{tg}$ is the unique element of $T^{(i)}$ such that $tg\overline{tg}^{-1} \in G^{(i+1)}$. With a family of such transversals $\{T^{(i)}\}$ and with $g \in \mathrm{Sym}(\Omega)$, either one can express g as a factored word, $g_{n-1}g_{n-2}\cdots g_1$, for $g_i \in T^{(i)}$, or else $g \notin G$.

§2 relies on the fact that one can efficiently construct a transversal from the strong generators $S^{(i)}$. The time for this construction depends on the data

structure for group membership being used. Specific data structures for group membership are defined at the beginning of §3.

The point α_i is called a *base point* relative to α if $|\alpha_i^{G^{(i)}}| \neq 1$. The sequence of all base points $B = (\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_b})$, with $i_1 < i_2 < \ldots < i_b$, is called an (ordered) *base* for G relative to α. The significance of a base is that each element g of G is uniquely determined by its base image $(\alpha_{i_1}^g, \alpha_{i_2}^g, \ldots, \alpha_{i_b}^g)$. Note that $b \leq \min(\log|G|, n-1)$. The size of a base may vary with the ordering α, but it is easy to show that two bases relative to two different orderings differ in size by at most a $\log n$ factor.

Finally, since many of the algorithms will be randomized ones, it is useful to review some terminology. A Monte Carlo algorithm is a randomized algorithm whose reliability (probability of success) can be increased at the cost of additional time. A Monte Carlo algorithm is Las Vegas, if it never returns an incorrect answer. (Hence, a Las Vegas algorithm may only return a correct answer or "don't know".) An algorithm is exponentially reliable if the probability of returning an incorrect answer or "don't know" is bounded above by $\exp(-n)$, for n the input parameter. A formal development of these definitions and their consequences is contained in the appendix.

Finally, for convenience, we will often say that a Monte Carlo algorithm runs in $O(f(n))$ time with *with fixed but arbitrarily high probability*. This is shorthand for saying that for any constant p with $0 < p < 1$, there is a constant c dependent on p such that if the algorithm runs for at most $cf(n)$ time, then it will return a correct answer with probability at least p.

2. Random Subproducts

It is known that efficient construction of random group elements would allow efficient randomized algorithms for many group constructions [**2, 22,29**]. Unfortunately in permutation groups, construction of truly random group elements usually depends on first constructing a strong generating set. (Babai has introduced an alternative approach toward approximately random group elements [**4**] which is sufficiently general to work over arbitrary black box groups.) This section introduces *random subproducts*, which can be efficiently constructed without knowing a strong generating set. Random subproducts were first defined for use in orbit calculations by Babai, Luks and Seress [**8**]. Their use in developing randomized algorithms for general group computations was first described in [**5**].

DEFINITION. Given a generating set $S = \{g_1, \ldots, g_r\}$ for a group G, a random subproduct on S is an element w of the form

$$w = g_1^{e_1} g_2^{e_2} \cdots g_r^{e_r},$$

where $e_i = 0$ or 1 with probability 1/2.

PROPOSITION 2.1. (from [**5, Lemma 2.2**]) *Let S generate a group G and let H be an arbitrary proper subgroup of G. Then a random subproduct w on S is not a member of H, with probability at least $1/2$.*

PROOF. Let $S = \{g_1, \ldots, g_r\}$. There is a largest i for which $g_i \notin H$ (since $H \neq G$). So, w can be decomposed into the form $w = ug_i^{e_i}v$, with $g_{i+1}, \ldots, g_r \in H$. Hence, $v \in H$.

We consider two cases. First, assume $u \in H$. With probability $1/2$, $e_i = 1$, and so $w = ug_iv \notin H$. In the second case, assume $u \notin H$. With probability $1/2$, $e_i = 0$, and so $w = uv \notin H$. This proves the proposition. □

The parameter L, an upper bound on the length of subgroup chains for a group G, plays an important role in applications of random subproducts. For permutation groups, Cameron, Solomon and Turull [**14**] have shown, using the classification of finite simple groups, a bound $L < 3n/2$ on the length of maximal subgroup chains. (Babai [**3**] had previously demonstrated a bound of $2n-3$ solely using elementary techniques.) For small base groups, $L = O^{\sim}(1)$. For subgroups of the general linear group $GL(n, q)$, there is an upper bound of $L = n^2 \log q$.

As an illustration of the use of random subproducts, consider an arbitrary finitely generated group $G = \langle S \rangle$ with upper bound L on the length of a subgroup chain. By using Proposition 2.1, we are able to show that for $c > 4$, cL random subproducts generate G with probability at least $1 - \exp(-(1 - 4/c)^2 cL/8)$. Hence, a generating set S can be replaced by a generating set of size cL in the time for $O(L|S|)$ group multiplications. A stronger version of this result is presented in §2.3.

From this initial idea of random subproducts, a series of strikingly improved randomized algorithms have been developed, that use little more than this basic idea and some combinatorics. (The parameter n is the permutation degree and S is the generating set. For simplicity, the generating sets are assumed to be of size $O(n)$ for the first three results, and the fourth result shows how to efficiently find such a generating set. The dependence on the size of the generating set and the reliability is given later in the paper.)

- §2.1: Elementary group membership: $O(n^4 \log^2 n)$ (exponential reliability)
- §2.2: Elementary normal closure: $O(n^3)$ (exponential reliability)
- §2.4: Normal closure using random prefixes: $O^{\sim}(n^2)$
- §2.3: Reduction of a generating set S to size $O(n)$: $O(|S|n \log n)$
- §2.3: Testing solvability of matrix groups over finite fields in polynomial time

The first two algorithms are based on simple extensions of random subproducts, *random Schreier subproducts* and *random normal subproducts*. In both cases, an element formed using these extensions has probability at least $1/4$ of enlarging the subgroup being incrementally constructed. The resulting chain of subgroups terminates in the target group. Random prefixes are a non-trivial variation of random subproducts.

The methods also apply to arbitrary black box groups. Black box groups

were first introduced by Babai and Szemerédi [**10**]. Informally, they can be described as a representation for elements of an abstract group for which group multiplication, group inverse, and recognition of the identity element can all be "effectively" computed. The black box versions are stated in terms of L, a bound on the length of chains of subgroups.

- §2.2: Elementary normal closure of H in G where H and G have generating sets of size s and t respectively: $O((s+t)L)$ multiplications and inverses
- §2.3: Reduction of a generating set S for G to size $O(L)$: $O(|S|\log L)$ multiplications
- §2.4: Normal closure using random prefixes where H and G are given by generating sets of size $O(L)$: $O(L\log^4 L)$ multiplications

2.1. Random Schreier Subproducts: Elementary Group Membership.

For groups $H \subseteq G$, a *transversal* of H in G is a set of coset representatives of H in G. Let G be a finitely generated group with a generating set $S = \{g_1, g_2, \ldots, g_s\}$, let $H \subseteq G$ be a subgroup of finite index, and let $T = \{t_1, \ldots, t_n\}$ be a transversal of H in G. For $g \in G$, let $\overline{g}$ be the unique element $t \in T$ such that $gt^{-1} \in H$. A *random Schreier subproduct* for H with respect to S and T is an instance of a product $(t_1 g \overline{t_1 g}^{-1})^{e_1}(t_2 g \overline{t_2 g}^{-1})^{e_2} \cdots (t_n g \overline{t_n g}^{-1})^{e_n}$ where g is a random subproduct of S, and the e_i are independent random variables uniformly distributed over $\{0, 1\}$. Note that a random Schreier subproduct requires at most $|S| - 1$ multiplies to compute the random subproduct g and at most $3(|T| - 1)$ additional multiplies to complete the computation, hence $O(|S| + |T|)$ multiplies overall. (We are assuming that each $\overline{t_i g}$ can be computed without any multiplies for each i, $1 \leq i \leq n$.) Random Schreier subproducts were first discovered in 1990, but have only recently appeared in print [**24**].

LEMMA 2.2. *Let $G = \langle S \rangle$ for finite S, and let $H \subseteq G$ be a subgroup of finite index. Let T be a transversal for H in G. Then a random Schreier subproduct h for H with respect to S and T can be computed at the cost of $O(|S| + |T|)$ group multiplications and inverses. Further, for any proper subgroup $K \subset H$,*

$$\text{Prob}(h \notin K) \geq 1/4.$$

PROOF. Let K be an arbitrary proper subgroup of H. Given a generating set $S = \{g_1, \ldots, g_s\}$ for G, there exists a maximal index i and a $\sigma \in T$, such that $\sigma g_i (\overline{\sigma g_i})^{-1} \notin K$. This follows from the fact that $K \subset H$ is a proper subgroup, and the Schreier generators generate all of H. By assumption, $\tau g_j (\overline{\tau g_j})^{-1} \in K$ for all $\tau \in T$ and $j > i$.

Given a random subproduct g on S, we show that with probability at least $1/2$ there exists a $\tau \in T$ such that $\tau g (\overline{\tau g})^{-1} \notin K$. Let u and v be prefixes and suffixes of g uniquely defined by $g = u g_i^{e_i} v$. The calculations below implicitly use the fact that $\overline{\overline{\tau}\, y} = \overline{\tau y}$ and hence $\overline{\tau} y (\overline{\overline{\tau} y})^{-1} = \overline{\tau} y (\overline{\tau y})^{-1}$. The argument divides into two cases. In the first case, $\tau u (\overline{\tau u})^{-1} \in K$ for all $\tau \in T$. So, with probability

$1/2$, $e_i = 1$, which implies $g = ug_iv$ (and so $\sigma u^{-1}g = \sigma g_i v$) and

$$\overline{\sigma u^{-1}}g(\overline{\sigma u^{-1}g})^{-1} = \underbrace{\overline{\sigma u^{-1}}u\sigma^{-1}}_{\in K}\ \underbrace{\sigma g_i(\overline{\sigma g_i})^{-1}}_{\notin K}\ \underbrace{\overline{\sigma g_i}v(\overline{\sigma g_i v})^{-1}}_{\in K} \notin K.$$

In the second case, there is a $\sigma' \in T$ such that $\sigma' u(\overline{\sigma' u})^{-1} \notin K$. So with probability $1/2$, $e_i = 0$, which implies $g = uv$ and

$$\sigma' g(\overline{\sigma' g})^{-1} = \underbrace{\sigma' u(\overline{\sigma' u})^{-1}}_{\notin K}\ \underbrace{(\overline{\sigma' u})v(\overline{\sigma' uv})^{-1}}_{\in K} \notin K.$$

Let h be a random subproduct on $\{\tau g(\overline{\tau g})^{-1}\colon\ \tau \in T\}$. If $\tau g(\overline{\tau g})^{-1} \notin K$ for some $\tau \in T$, then $h \notin K$ with probability at least $1/2$. Since the exponents $\{e_i'\} \subseteq \{0,1\}$ for the random subproduct h are chosen independently of the $\{e_i\}$ for g, there is an overall probability of at least $1/4$ that $h \notin K$. □

An important application of Lemma 2.2 occurs in efficiently finding generators of $H \subseteq G$ when only generators for G and a transversal of H in G are known. This is the case in finding generators of a point stabilizer subgroup. The proof of reliability requires Chernoff's bound [**15**].

Chernoff's Bound. *Let S_t be a random variable equal to the number of successes in t independent Bernoulli trials in which the probability of success is p $(0 < p < 1)$. Let $0 < \epsilon < 1$. Then*

$$\text{Prob}(S_t \leq \lfloor (1-\epsilon)pt \rfloor) \leq e^{-\epsilon^2 pt/2}.$$

Theorem 2.3. *Let G be a (possibly infinite) group with finite generating set S, let L be an a priori upper bound on the length of subgroup chains in G, and let T be a transversal for $H \subseteq G$. Given a constant $c > 4$, one can construct a set of cL generators for H with probability at least $1 - \exp(-(1-4/c)^2 cL/8)$ using at most $cL(|S| + 3|T| - 4)$ group multiplications and inverses.*

Proof. We will construct a set $\{h_1, \ldots, h_{cL}\}$ of random Schreier subproducts, which generates H with the stated probability. The construction of each random Schreier subproduct takes at most $|S| + 3|T| - 4$ group operations and so the bound on the number of group operations follows directly. Define the generation of each h_i as a trial, and define the event E_i as the condition that either $h_i \notin \langle\{h_1, \ldots, h_{i-1}\}\rangle$ or $\langle\{h_1, \ldots, h_{i-1}\}\rangle = H$. Then by Lemma 2.2, $\text{Prob}(E_i) \geq 1/4$ for all i. Note that $\text{Prob}(H = \{h_1, \ldots, h_{cL}\})$ is bounded below by the probability that E_i holds for at least L distinct i. If the generation of each h_i is a trial, and the event E_i is a "success", then this probability is in turn bounded below by the probability of at least L successes in cL Bernoulli trials with parameter $p = 1/4$. Setting $t = cL$, $p = 1/4$, and $\epsilon = 1 - 4/c$ in Chernoff's bound, we can estimate the probability as $\text{Prob}(S_t > L) \geq 1 - \exp(-(1-4/c)^2 cL/8)$. □

For permutation groups, G, of degree n, one can apply Theorem 2.3 to finding generators of a point stabilizer group, G_x. For this case, $|T| \leq n$ and $L = 3n/2$ [**14**], yielding the following corollary.

COROLLARY 2.4. *Let G be a permutation group of degree n, generated by S. Given $c > 4$, one can construct a set of cn generators for a point stabilizer subgroup of G with probability at least $1 - \exp(-(1-4/c)^2(3/16)cn)$ using at most $(3/2)cn(|S| + 3n - 4)$ group multiplications and inverses.*

An algorithm to construct the full point stabilizer sequence can be derived through $n-1$ applications of Corollary 2.4. In the following procedure, the subroutine `First-point-moved` is called with a set U of permutations and returns either the first point $j < n$ moved by some element of U or n if each element of U is the identity permutation.

`Procedure Construct-SGS`
Input: A generating set S for G
Output: A list $(S^{(i_1)} \dots, S^{(i_m)})$ of generating sets for the point stabilizer sequence of G where $i_1 < i_2 < \ldots < i_m$ and $S^{(i_j)}$ generates $G^{(i_j)}$

Let $\beta_1 \leftarrow$ `First-point-moved`(S)
If $\beta_1 = n$ then return()
Else set $S^{(\beta_1)} \leftarrow S$
Set $j \leftarrow 1$
Loop:
  Let T be a transversal for $(\langle S^{(\beta_j)} \rangle)_{\beta_j}$ in $S^{(\beta_j)}$
  Let U be a set of $18n$ random Schreier subproducts
    with respect to $S^{(\beta_j)}$ and T
  Let $\beta_{j+1} \leftarrow$ `First-Point-Moved`(U)
  If $\beta_{j+1} = n$ then return$(S^{(\beta_1)}, \dots, S^{(\beta_j)})$
  Else set $S^{(\beta_{j+1})} \leftarrow U$, set $j \leftarrow j+1$, and goto Loop

COROLLARY 2.5. *Given a permutation group $G = \langle S \rangle$ of degree n, procedure* `Construct-SGS` *will return a strong generating set for G in time $O(n^4 + n^2|S|)$ with probability at least $1 - \exp(-n)$.*

PROOF. The proof is by repeated application of Corollary 2.4. The time to compute the first transversal and the $18n$ random Schreier subproducts with respect to $S = S^{(\beta_1)}$ is $O(n^2|S| + n^3)$, and $|S^{(\beta_2)}| = O(n)$. The time for each later iteration is $O(n^3)$. The probability that $\langle S^{(\beta_i)} \rangle = G^{(\beta_i)}$ is at least $1 - \exp(-2n)$ by application of Corollary 2.4 with $c = 18$. The probability that this will hold for all β_i is at least $(1 - \exp(-2n))^n \geq 1 - n\exp(-2n) = 1 - \exp(\log(n) - 2n) \geq 1 - \exp(-n)$. □

REMARK. *The procedure* `Construct-SGS` *can be upgraded to an $O(n^4)$ Las Vegas group membership algorithm by applying the strong generating test presented in* [**17**].

2.2. Random Normal Subproducts: Elementary Normal Closure.

In applications to group membership, one often needs only to find generators (not necessarily strong) of a normal closure. In this case, algorithms that

are faster than group membership are available [**5**]. An elementary algorithm ($O(n^3)$) is presented first, followed by a more complicated $O(n^2 \log^4 n)$ algorithm in §2.4. The $O(n^3)$ normal closure algorithm uses the idea of incremental subgroup construction described in §2. However, instead of random subproducts, it uses *random normal subproducts*. Random normal subproducts were discovered in 1990, but have not yet appeared in print. Since the techniques described here apply to black box groups, the discussion will be formulated in terms of a known upper bound L for the length of chains of subgroups.

The problem of constructing strong generators for a normal closure is distinct from finding any generating set, and has long had an unclear relation to the problem of group membership. Our recent work with Luks [**20**], discussed in §5, showed that for permutation groups, normal closure can be efficiently reduced to group membership. In particular, construction of strong generators of a normal closure requires $O^{\sim}(n^3)$ time in general and $O^{\sim}(n)$ time for a small base group.

Let $H \subseteq G$, with $H = \langle U \rangle$ and $G = \langle S \rangle$. A *random normal subproduct with respect to U and S* is a conjugate of the form h^g where h is a random subproduct on U and g is a random subproduct on S.

LEMMA 2.6. *Let $H = \langle U \rangle$ and $G = \langle S \rangle$ be subgroups of a (possibly infinite) group K Suppose that H is not normalized by G. Let h^g be a random normal subproduct with respect to U and S. Then*

$$\text{Prob}(h^g \in \langle H^G \rangle \setminus H) \geq 1/4.$$

PROOF. By hypothesis, the normalizer $N_G(H)$ of H in G is a proper subgroup of G. Hence if g is a random subproduct on S, then $\text{Prob}(g \notin N_G(H)) \geq 1/2$ by Lemma 2.2. So, $H^g \cap H$ is proper in H^g with probability at least $1/2$. Let h be a random subproduct on $U = \{h_1, h_2, \ldots, h_t\}$. The random normal subproduct $h^g = ({h_1}^{e_1} \cdots {h_t}^{e_t})^g = ({h_1}^g)^{e_1} \cdots ({h_t}^g)^{e_t}$, h^g can also be viewed as a random subproduct on $\{{h_1}^g, \ldots, {h_t}^g\}$, a generating set for H^g. Note that the choice of the random $\{e_1, \ldots, e_t\} \subseteq \{0,1\}$ is independent of those chosen for g. Given that $H^g \cap H$ is proper in H^g, it follows again from Lemma 2.2 that $\text{Prob}(h^g \notin H) = \text{Prob}(h^g \notin H^g \cap H) \geq 1/2$. Combining this with the probability $\text{Prob}(H^g \cap H \neq H^g) \geq 1/2$ yields $\text{Prob}(h^g \notin H) \geq 1/4$. □

THEOREM 2.7. *Let H and G be groups with $H = \langle U \rangle$ and $G = \langle S \rangle$, and let L be an a priori upper bound on the length of a chain of subgroups of $\langle H^G \rangle$. Let $c > 4$ be a given constant. Then one can construct a set of cL generators for $\langle H^G \rangle$ with probability at least $1 - \exp(-(1 - 4/c)^2 cL/8)$ using at most $cL(|S| + |U| + (cL + 1)/2)$ group multiplications and inverses.*

PROOF. Let $U' = U$ initially. Let h^g be a random normal subproduct with respect to U' and S, and then augment U' to $U' \cup \{h^g\}$. Repeat the process cL times. By Lemma 2.6, while $\langle U' \rangle$ is not normalized by G, each new random normal subproduct added to U' has probability at least $1/4$ of enlarging the subgroup $\langle U' \rangle$. The proof now follows as in Theorem 2.3. □

The following corollary uses the fact that if b is the size of an arbitrary base for a permutation group G of degree n and L is the length of a maximal subgroup chain in G, then $L \leq b \log n$. (This can be easily shown by first noting that $L \leq \log(|G|)$ and then observing that $|G| \leq n^b$, hence $log(|G|) \leq b \log(n)$.)

COROLLARY 2.8. *Let H and G be permutation groups of degree n with $H \subseteq G$ and let G have a base of size b. Assume that both H and G are given by generating sets of size at most $b \log n$. Then for a given constant $c > 4$, a generating set of size at most $cb \log n$ for $\langle H^G \rangle$ can be obtained with probability at least $1 - 1/n^{kb}$ for $k = c(1-4/c)^2(\log_2 e)/8$, using $cb^2 \log^2 n$ group multiplications and inverses. In particular, if G is a small base group, then the algorithm runs in time $O^{\sim}(n)$.*

2.3. Reduction of Generators.

Reduction of generators to a set of size $O(L)$ is an important principle when an algorithmic construction yields too many generators. Two examples are presented to illustrate this principle: an elementary group membership algorithm which runs in time $O^{\sim}(n^4)$; and a polynomial time test for solvability and nilpotence of a matrix group defined over a finite field.

The proof of the result on reduction of generators uses a *random subproduct on S of length ℓ*, a random subproduct on a sequence of of length ℓ, with elements randomly drawn (with replacement) from S. The proof is based on ideas first presented in [**5, Theorem 2.3**]. It avoids the explicit computation of one of the constants for purposes of brevity.

THEOREM 2.9. (from [**5, Theorem 2.3**]) *Let $G = \langle S \rangle$ be a finite group. Let L be a known upper bound on the length of all subgroup chains in G. Then for any fixed parameter p such that $0 < p < 1$, with probability at least p one can find a generating set S' with $|S'| = O(L \log(1/(1-p)))$, using $O(|S| \log L \log(1/(1-p)))$ group operations.*

PROOF. Let $|S| = s$. Assume $s > 4L$ or else there is nothing to do. Starting from $S' = \emptyset$, successively place in S', cL random subproducts on S, each of length s/L, for some $c > \lceil 4e \rceil$. We claim that after this initial phase, with probability at least $1 - e^{-cL/(16e)} > 1 - e^{-c/(16e)}$, fewer than L of the elements of S are outside of $\langle S' \rangle$. (e is the base of the natural logarithm.)

The claim can be rephrased in the language of indicator functions. Let $g_1, g_2, \ldots, g_{cL}$ denote the random subproducts on S produced by the procedure. Let ξ_i be the indicator variable taking value $\xi_i = 1$ if $g_i \notin \langle g_1, \ldots, g_{i-1} \rangle$ or if $|S \setminus \langle g_1, \ldots, g_{i-1} \rangle| < L$. Otherwise, set $\xi_i = 0$. Note that by the definition of L, $\sum_{i=1}^{cL} \xi_i > L \Rightarrow |S \setminus \langle g_1, \ldots, g_{cL} \rangle| < L$.

Next, we compute a uniform lower bound for $\text{Prob}(\xi_i = 1)$, independent of i and $\{g_1, \ldots, g_{i-1}\}$. If $|S \setminus \langle g_1, \ldots, g_{i-1} \rangle| \geq L$, then at least L of the s elements of S are not in $\langle g_1, \ldots, g_{i-1} \rangle$. So, a random sequence on S (with replacement) of length s/L contains at least one element of $S \setminus \langle g_1, \ldots, g_{i-1} \rangle$ with probability at least $1 - (1 - L/s)^{s/L} > 1/e$. Since g_i is a random subproduct on S of length s/L, by Proposition 2.1 $g_i \notin \langle g_1, \ldots, g_{i-1} \rangle$ with probability at

least $1/(2e)$. Thus, in the case that $|S \setminus \langle g_1, \ldots, g_{i-1}\rangle| \geq L$, $\text{Prob}(\xi_i = 1) = \text{Prob}(g_i \notin \langle g_1, \ldots, g_{i-1}\rangle) > 1/(2e)$. In the other case, if $|S \setminus \langle g_1, \ldots, g_{i-1}\rangle| < L$, then $\text{Prob}(\xi_i = 1) = 1$. Hence, $\text{Prob}(\xi_i = 1) > 1/(2e)$ for all i, independently of $\{g_1, \ldots, g_{i-1}\}$.

This implies $\text{Prob}(\xi_i = 1 \mid \xi_1 = a_1, \ldots, \xi_{i-1} = a_{i-1}) > 1/(2e)$ independently of $a_1, \ldots, a_{i-1}$. So, $\text{Prob}(\sum_{i=1}^{cL} \xi_i \geq L)$ is bounded below by the probability of at least L successes in cL Bernoulli trials with probability parameter $p' = 1/(2e)$. Therefore one can apply Chernoff's bound (§2.1). Taking $\epsilon = 1 - 1/(cp')$, $t = cL$ and $S_t = \sum_{i=1}^{cL} \zeta_i$, we have

$$\text{Prob}(|S \setminus \langle S'\rangle| < L) \geq \text{Prob}(\sum_{i=1}^{cL} \xi_i > L) \geq 1 - e^{-\epsilon^2 p' t/2}.$$

For $c > 4e$, $\epsilon > 1/2$ and so $\text{Prob}(|S \setminus \langle S'\rangle| < L) \geq 1 - e^{-cL/(16e)} \geq 1 - e^{-c/(16e)}$. Note that as c goes to infinity, the lower bound on the last probability goes to 1, independently of s and L.

After this initial phase, we choose a new constant c' and continue for $\log_2 L$ more rounds. After round $i-1$ assume that $|S - \langle S'\rangle| < L/2^{i-1}$. The set S' will be augmented by $c'L/2^i$ products of random sequences of S of length $2^{i-1}s/L$ during round i. We shall show that with a suitable reliability, $|S - \langle S'\rangle| < L/2^i$ after round i. If $|S - \langle S'\rangle| < L/2^i$ already after round $i-1$, nothing need be proved. Otherwise, $L/2^i \leq |S - \langle S'\rangle| < L/2^{i-1}$, and define $\tilde{n} = 2^{i-1}s/L$, $\tilde{p} = |S - \langle S'\rangle|/s \geq L/(2^{i-1}s)$, and $\tilde{q} = 1 - \tilde{p} > (1 - L/(2^{i-1}s))$. So, a random sequence of S (with replacement) of length $\tilde{n} = 2^{i-1}s/L$ will include exactly one element of $S - \langle S'\rangle$ with probability $\tilde{n}\tilde{p}\tilde{q}^{\tilde{n}-1} > (2^{i-1}s/L)\,(L/(2^i s)) \times (1 - L/(2^{i-1}s))^{(2^{i-1}s/L)-1} > (1/2e)\,(1 - L/(2^{i-1}s))^{-1} > 1/(2e)$. If a product of the elements of the random sequence is added to S', then $|S - \langle S'\rangle|$ is diminished by at least one with probability at least $1/(2e)$. So we form $c'L/2^i$ such random sequences and add the corresponding products to S'. For a sufficiently large constant c', if $|S - \langle S'\rangle| < L/2^{i-1}$ was true after round $i-1$, then $|S - \langle S'\rangle| < L/2^i$ in round i, with probability at least $1 - 2^{-L/2^{i+1}}$.

After round number $\log_2 L$, $|S - \langle S'\rangle| = 0$ and S' generates G. The probability of error on any round subsequent to the initial one is at most $\sum_{i=1}^{\log_2 L} 2^{-L/2^{i+1}} < 1/2$. Finally, it is clear from Chernoff's bound that any fixed probability p of the theorem can be achieved by always taking a multiple of $\log_2(1/(1-p))$ times as many random subproducts and products over random sequences as indicated in the proof. □

REMARK. *The algorithm in the proof of Theorem 2.9 is Monte Carlo, since any probability p of correctness can be achieved at the cost of extra time.*

In application of Theorem 2.9 to the construction of subgroup chains such as the point stabilizer sequence or the commutator series, each subgroup in the chain is usually constructed using a generating set for its predecessor in the chain together with some defining condition. Without taking care to reduce the size of

the generating set used for the construction, it is possible to have an exponential blowup in the number of generators as one proceeds down the subgroup chain.

As a first illustration of the importance of reduction of generators, we sketch an elementary $O(n^3 k \log^2 n)$ group membership algorithm [**5**], where k is the maximum orbit length. The algorithm consists of at most n point stabilizer constructions according to the algorithm described in Corollary 2.10 below. In this case, $L < 3n/2$. The algorithm associated with Corollary 2.10 consists of forming Schreier generators (of which there are at most nk), and then applying Theorem 2.9 to find $O(n)$ generators of the point stabilizer subgroup.

COROLLARY 2.10. (from [**5, Corollary 2.8**]) *Let $G = \langle S \rangle$ be a permutation group of degree n. Assume $|S| = O(n)$. Let $x \in \Omega$. Then for any fixed parameter p such that $0 < p < 1$, with probability at least p one can find a set of $O(n \log(1/(1-p)))$ generators of the point stabilizer subgroup G_x in Monte Carlo time $O(n^2 k \log n \log(1/(1-p)))$ where k is the length of the G-orbit of x.*

To achieve the elementary group membership, one forms the $O(n)$ point stabilizer subgroups using Corollary 2.10 with $p = 1 - 1/(2n)$. Each point stabilizer requires $O(n^2 k \log^2 n)$ time and achieves a reliability of at least $1 - 1/(2n)$. Thus, one achieves a strong generating set in $O(n^3 k \log^2 n)$ Monte Carlo time with reliability at least $1/2$.

A second illustration of the use of reduction of generators is a Monte Carlo polynomial time test for solvability and nilpotence of a matrix group $G = \langle S \rangle$ with S a set of $n \times n$ matrices over $GF(q)$. The crude bound $L = n^2 \log(q)$ suffices. The algorithm alternates between reducing the size of the generating set and computing generators for the next group in the derived series or descending central series via the normal closure algorithm. The reduction of generators employs probability of success $p = 1 - 1/(2L)$. After at most L iterations, if each of the resulting generators is the identity, then the test is satisfied and the algorithm asserts that the group is solvable or nilpotent respectively. Constants can be chosen so that the probability of error is most some constant p. Since each iteration takes polynomial time in n and $\log q$ and since there are at most L iterations with $L = n^2 \log q$, the combined test is polynomial in n and $\log q$. This gives rise to the following proposition.

PROPOSITION 2.11. (from [**5, Corollary 1.4**].) *A subgroup of $GL(n, q)$ can be tested for solvability and nilpotency by a Monte Carlo algorithm in the time for $O(L^3 \log^2 L)$ multiplications and inverses, with $L = n^2 \log q$.*

The estimate of $L = n^2 \log q$ is asymptotically tight. To see this, note that if $q = p^n$, for p a prime, then a Sylow p-subgroup of $GL(n, q)$ has order $p^{(\log_p q) n(n-1)/2}$. Hence for p fixed, only r varies, and $L = (\log_p q) n(n-1)/2 = \Omega(n^2 \log q)$.

2.4. Random Prefixes: Normal Closure Revisited.

An alternative normal closure algorithm which runs in time $O^{\sim}(n^2)$ for permutation groups of degree n will now be discussed. The key idea is the use of random prefixes to produce new elements for the group under construction.

DEFINITION. A *random prefix* of a sequence of group elements $(g_1, g_2, \ldots, g_k)$ is an instance of a product of the first i elements, for i a random integer from 1 through k.

To see how random prefixes are used in the normal closure algorithm, assume that the maximal length of a chain of subgroups of G is L and that G and H are each given by generating sets of size $O(L)$. For a suitably large m ($m = \Theta(L \log^2 L)$), form lists S_G and T of length m initialized with the generating sets for G and H respectively and padded to the end with the identity element. Let $r = \Theta(\log(m))$ and let $\pi_1, \ldots, \pi_r$ be random permutations of $\{1, \ldots, m\}$ chosen at the beginning of the computation. Apply randomly chosen elements π_i and π_j to S_G and T respectively, let g be a random prefix of $S_G^{\pi_i}$ and h a random prefix of T^{π_j}. Replace one of the original identity elements of T by h^g. Then with fixed but arbitrarily high probability, after $O(mr^2)$ rounds, the list T will contain generators for $\langle H^G \rangle$. In §2.3, we showed how to reduce the generating set T to one of size $O(L)$.

The proof of correctness depends on being able to "spread out" the distance between two arbitrary elements in the list S_G or T using some permutation in S_m. A set of permutations having this property is called a set of ϵ-spreaders. The proof shows that the set of random permutations $\{\pi_1, \ldots, \pi_r\} \subseteq S_m$ is in fact a set of ϵ-spreaders. This relies on the construction of special permutations in S_m, called *ϵ-spreaders*. Given an arbitrary subset of generators, at least one of $\log L$ ϵ-spreaders is guaranteed to permute a list of m generators, so that the sum of distances of each element of the arbitrary subset to its nearest neighbor from the same subset is bounded below by ϵ. Efficient deterministic construction of ϵ-spreaders remains an important open question. An answer would allow a modified normal closure algorithm achieving both exponential reliability and a smaller time complexity.

THEOREM 2.12. (from [**5, Theorem 3.5**]) *Let G and $H \subseteq G$ be finite groups and assume all subgroup chains in G have length at most L. Assume further that G and H are given by generating sets of size $O(L)$. Then one can construct $O(L)$ generators for $\langle H^G \rangle$ with fixed but arbitrarily high probability using $O(L \log^4 L)$ group operations.*

COROLLARY 2.13. (from [**5, Theorem 3.5**]) *Let H and G be permutation groups of degree n with $H \subseteq G$. Assume that both H and G have generating sets of size at most $O(n)$. Then a generating set of size $O(n)$ for $\langle H^G \rangle$ can be obtained with fixed but arbitrarily high probability using $O(n \log^4 n)$ group multiplications and inverses. If G is a small base group ($\log |G| = O^{\sim}(1)$), then $O^{\sim}(1)$ such group operations are required.*

3. Nearly Optimal Group Membership

This section reviews our work on "nearly optimal" group membership algorithms. The general group membership algorithm [**5**] operates in $O^{\sim}(n^3)$ time for generating sets of size $O(n)$. It is nearly optimal in the sense that its time approaches the time for Gaussian elimination. Gaussian elimination for matrices over a finite field is in some sense equivalent to a special case of this algorithm, and Gaussian elimination requires $O(n^3)$ time. The more specialized group membership algorithm for small base groups [**6**] operates in $O^{\sim}(n)$ time for generating sets of size $O^{\sim}(n)$. The time $O^{\sim}(n)$ is the same as the time to read the input, in the sense of "soft O", and so is considered nearly optimal for small base groups. Finally, a novel, elementary group membership algorithm is described, which duplicates the "near optimality" achieved [**BCFLS**] with a small gain in complexity.

The nearly optimal algorithms are all Monte Carlo. However, an exponentially reliable (or Las Vegas or deterministic) algorithm continues to elude us. This is an important open question, which would be solved by either an exponentially reliable or Las Vegas strong generating test operating in $O^{\sim}(n^3)$ time. (Note that a deterministic $O^{\sim}(n^3)$ strong generating test brings no special advantage, since, when combined with a Monte Carlo $O^{\sim}(n^3)$ group membership algorithm, it would yield a new group membership algorithm that is still only Las Vegas.) To date, we have presented a $O(|S|n^3)$ deterministic strong generating test [**17**], and Babai, Luks and Seress have also derived a $O^{\sim}(|S|n^3)$ deterministic strong generating test [**8**].

3.1. Efficient Data Structures for Schreier Trees and Group Membership.

Let $G = \langle S \rangle$ be a permutation group acting on $\Omega = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. *Schreier trees* are often used for their space-efficiency in storing coset representatives of G_α in G for $\alpha \in \Omega$.

A *Schreier tree* for $G^{(i)}$ is a directed labeled tree $\mathcal{T}$ with root node α_i, nodes from $\alpha_i^{G^{(i)}}$, and edge labels chosen from a label set of permutations $S \subseteq G^{(i)}$. A label g for the directed edge (β, γ) satisfies $\beta^g = \gamma$. Coset representatives are obtained by taking the product of edge labels along a path from the root to the specified node. A *Schreier vector data structure* for G relative to the ordering α is a sequence $\{\mathcal{T}_i\}_{i=1}^n$ of *Schreier trees*, where $\mathcal{T}_i$ is a Schreier tree relative to $G^{(i)}$. $\mathcal{T}_i$ is *complete* if the set of nodes of T_i coincides with $\alpha_i^{G^{(i)}}$ and the Schreier vector data structure is *complete* if $\mathcal{T}_i$ is complete for $1 \leq i \leq n$. Schreier trees were first discovered by Sims [**35**], where they were called Schreier vectors. We have revised the terminology in keeping with more recent computer science concepts.

The permutation g is said to *sift through* $\{\mathcal{T}_i\}_{i=1}^n$ if we can write g in the form

$$g = g_{n-1} g_{n-2} \cdots g_1,$$

where g_i is a coset representative obtained from $\{\mathcal{T}_i\}$ (using the procedure described above) and $\alpha_i^{gg_1{}^{-1}\cdots g_{i-1}{}^{-1}} = \alpha_i^{g_i}$. The number of multiplications and inverses required to sift an element is proportional to the sum of the depths of the Schreier trees. In the case where the Schreier vector data structure is complete, the above factorization has two important implications. The first is the ability to generate random elements of G according to the uniform distribution. This is achieved by creating an element g whose factorization has the above form with the elements g_i chosen at random according to the uniform distribution from amongst the cosets for $G^{(i+1)}$ in $G^{(i)}$ defined by $\mathcal{T}_i$. The second implication is a test for membership in G of an arbitrary permutation. The test consists of attempting to factor an arbitrary permutation through $\{\mathcal{T}_i\}_{i=1}^n$. The factorization will succeed if and only if the element belongs to G. This was Sims's original group membership test [**35**].

Efficient Schreier trees must satisfy two goals. They must use a relatively small number of labels from G. This becomes critical as n approaches 1,000,000. Second, they must have shallow depth, so as to make recovery of a coset representative efficient.

It has been argued that Schreier trees built up by breadth-first search on some generating set are usually shallow. Nevertheless, there are important counterexamples. For example. just consider a Schreier tree for the cyclic group Z_n with one generator, for n large. Another example is $PSL(2,q)$, acting naturally on the points of the projective line, where the last point stabilizer subgroup is cyclic of order $(q-1)/(2,q-1)$. Another natural example is the symmetric group with generating set of size $n-1$, $\{(1\ 2), (2\ 3), \ldots, (n-1\ n)\}$.

3.1.1. *Short Schreier Trees.*

In the special case when random elements are available, one can build *short Schreier trees* with $O(\log n)$ labels and $O(\log n)$ depth. This is always the case, in the context of a base change, and was the basis for a fast randomized base change algorithm [**18, 22**]. The following key observation was discovered independently, but variations of it have appeared from as early as 1965 [**1, 7, 25**].

Let G act transitively on Ω. For fixed $P \subseteq \Omega$ and random $g \in G$,

$$E(|P^g - P|) = |P|\ |\Omega \backslash P|/n.$$

We identify P with the current nodes of a partially built Schreier tree. As long as fewer than half of the elements of Ω are in the tree, each random element has a fixed positive probability of expanding the tree by a constant factor. Thereafter, with fixed positive probability, each random element will decrease by a constant factor the number of nodes of Ω not yet in the tree.

This observation is the basis of the following theorem.

PROPOSITION 3.1. (from [**22, Theorem 3.5**]) *Let $\mathcal{O} = \alpha^G$. Then there exists a constant $c > 0$ (we can take $c = 21$) such that for all $\delta \geq 1$, $\lceil \delta c \log_2 |\mathcal{O}| \rceil$ random group elements suffice to build a short Schreier tree, with probability at least $1 - 1/|\mathcal{O}|^{2\delta}$.*

It is still an open problem to deterministically build short Schreier trees (depth $O(\log n)$) efficiently. This would eliminate the need for a source of random elements, and possibly further improve the time. We have almost accomplished this goal in the important class of *small base* permutation groups, as discussed in the next section.

3.1.2. *Cube Schreier Trees.*

Cubes [**10**] were originally invented by Babai and Szemerédi in the theoretical context of black box groups. A constructive adaptation was provided in [**6**] for building *cube Schreier trees* (Schreier trees of depth $O(\log |G|)$ with $O(\log |G|)$ labels). The construction is deterministic, and also is the basis for a fast deterministic cyclic base change [**19**] discussed in §4.

Given a sequence $R = (g_1, \ldots, g_r)$, define the cube

$$C(R) = \{g_1^{e_1} g_2^{e_2} \cdots \ g_r^{e_r} : e_i \in \{0, 1\}\}.$$

It is easy to see that for $g \in G$

$$g \notin C(R)^{-1}C(R) \iff |C(R \cup \{g\})| = 2|C(R)|.$$

Choosing such a g is made efficient by noting that

$$\alpha_1^g \notin \alpha_1^{C(R)^{-1}C(R)} \Rightarrow g \notin C(R)^{-1}C(R).$$

Thus, while $\alpha_1^{C(R)^{-1}C(R)} \neq \Omega$, we can double the size of the cube $|C(R)|$ by appending a g_{r+1} such that $\alpha_1^{g_{r+1}} \notin \alpha_1^{C(R)^{-1}C(R)}$. After at most $\log_2 |G|$ such steps, $\alpha_1^{C(R)^{-1}C(R)} = \Omega$ and $length(R) \leq \log_2 |G|$. A Schreier tree with depth at most $2 \log_2 |G|$ can then be built using labels $R^{-1} \cup R$ in the obvious way. Any Schreier tree with upper bound $2 \log_2 |G|$ on its depth and the number of labels is a *cube Schreier tree*. If one has a cube Schreier tree for each $G^{(i)}$ for $1 \leq i \leq n-1$, then the family of such trees is a *cube Schreier vector data structure*.

The cube Schreier trees have important implications for implementations. They were shown [**6**] to be one of the keys to a $O^{\sim}(n)$ algorithm for group membership. However, for efficient implementation, the obvious construction can be modified to the pseudo-code below, while preserving the same asymptotic time and space complexity. As compared to a breadth-first search to building Schreier trees, the cube approach below can often be faster in implementations by selectively adding new group elements that are guaranteed to at least double the size of the cube. At the same time, it provides a theoretical guarantee against very deep Schreier trees. This can be illustrated if the reader applies the following code to the case of Z_n with a single generator.

Procedure *Build-Cube-Schreier-Tree*(S, x)
Input: A generating set S for G and point $x \in \Omega$.
Output: A cube Schreier tree $\mathcal{T}$ for x^G with the property that $\texttt{Labels}(\mathcal{T}) = R \cup R^{-1}$, where $C(R)$ is a non-degenerate cube and $\texttt{depth}(\mathcal{T}) \leq 2|R|$.

Initialize $R \leftarrow ()$

Set $\texttt{root}(\mathcal{T}) \leftarrow \{x\}$, $\texttt{Labels}(\mathcal{T}) \leftarrow ()$
  While there exists $g \in S$ such that $\texttt{Nodes}(\mathcal{T})^g \neq \texttt{Nodes}(\mathcal{T})$ do
    Let $y \in \texttt{Nodes}(\mathcal{T})$ such that $y^g \notin \texttt{Nodes}(\mathcal{T})$
    Let $h =$`Coset-rep`$(\mathcal{T}, y)$
    Append hg to R
    Build a new $\mathcal{T}$ using breadth first search with $R \cup R^{-1}$ to level $2|R|$
Return$\{\mathcal{T}\}$

PROPOSITION 3.2. (from [**19, Proposition 2.3**]) *Procedure Build-Cube-Schreier-Tree is correct and takes time* $O(n \log^2 |G| + n|S| \log |G|)$.

3.1.3. *Local Expansion (Babai).*

Babai's "local expansion lemma" [**4**] is the basis for an efficient randomized strong generating test for permutation groups [**6**], which requires only $O^{\sim}(|S|n)$ time for small base groups with generators S. As with most strong generating tests, when the generating set is not strong, the algorithm exhibits an element that does not sift through a group membership data structure based on the current generating set.

LEMMA 3.3. (from [**4, Local Expansion**]) *Let* $G = \langle S \rangle$, *and let* $T = S \cup S^{-1} \cup \{e\}$. *Let* T^t *be the set of* t*-term products of* T. *Then any subset* $D \subseteq T^t$ *of a group* G *satisfying* $|D| \leq |G|/2$ *also satisfies*

$$|Dg \backslash D| \,/\, |D| \geq 1/(4t)$$

for at least one generator $g \in S$.

Babai and Szegedy [**9**] recently found an inequality,

$$|DT \backslash D| \,/\, |D| \geq 2/(2t+1),$$

under suitable hypotheses, that is superior in many applications. In addition, by replacing the original hypothesis by the requirement that the diameter of the Cayley graph G with generators S be strictly larger than the diameter of the embedded subgraph D, they achieve the improved inequality [**9**],

$$|DT \backslash D| \,/\, |D| \geq 2/(t+2).$$

The principle of the strong generating test is to identify D with all factored words of G in S according to some group membership data structure such as Schreier trees. Assuming a base of size b, and assuming Schreier trees of depth f, t can be bounded by $t \leq bf = O(\log^c n)$. So if $|D| \leq |G|/2$, there is a $g \in S$ such that for random $d \in D$, $dg \notin D$ with probability at least $1/4t$. Alternatively, if $|D| \leq |G|/2$, for random $g \in S$ and $d \in D$, $dg \notin D$ with probability at least $1/(4t|S|)$. This test is repeated for each point stabilizer subgroup $G^{(i)}$ and strong generating set $G^{(i)} \cap S$.

Random subproducts further improve the strong generating test. Instead of testing if $dg \in D$ for random $d \in D$ and for each $g \in G$, it suffices to test if $dw \in D$ for w a random subproduct of the generating set S. If $|D| \leq |G|/2$,

then $dw \notin D$ with probability at least $1/(64t)$ [**6, Lemma 3.4**], which can be improved to $1/(16t)$ using the newer constant for local expansion. This result should be compared with the probability of at least $1/(4t|S|)$ in the previous paragraph.

3.2. Group Membership: Large and Small Base.

From a complexity viewpoint in computational group theory, group membership is probably the problem most studied. Originally, in practical implementations, the formal algorithms were not competitive with a straightforward heuristic algorithms such as random Schreier Sims [**29**]. Using the new ideas described here, and in collaboration with Babai, Luks and Seress, we now have competitive Monte Carlo implementations for both the large and small base case based on new algorithms with provable complexities [**5, 6**]. (The timing in the first theorem is revised from the original.)

THEOREM 3.4. (from [**5, Theorem 4.1**]) *Given $O(n^2)$ generators of a finite permutation group G acting on n points, one can construct a strong generating set for G in $O(n^3 \log^4 n)$ Monte Carlo time. The space requirement of the procedure is $O(n^2 \log^2 n)$.*

THEOREM 3.5. (from [**6, Theorem 6.2**]) *Let $G = \langle S \rangle \leq Sym(\Omega)$ be a group with $|\Omega| = n$, and suppose that b is the maximal size of a non-redundant base. Then there is a Monte Carlo algorithm that returns a strong generating set for the case of transitive G in $O(n \log^3 |G| + b^3 (\log b)(\log^3 |G|)(\log n) + n|S| \log |G|)$ time. For intransitive G, an extension of the algorithm returns a strong generating set in $O(n \log^3 |G| + nb^2 \log^2 |G| \log(b + \log n) + b^3 (\log b)(\log^3 |G|)(\log n) + n|S| \log |G|)$ time. The constructed strong generating set supports membership testing in $O(n \log |G|)$ time and the memory requirement is $O(nb \log |G| + n|S|)$ during the construction.*

We have written straightforward implementations of about one to two thousand lines for both algorithms in our LISP system. Seress and Weisz have done the same in C, and report on their experience in this proceedings [**34**]. At this time, Leon's heuristic implementation [**29**] of randomized Schreier Sims is usually faster, but does not provide a theoretical guarantee of correctness under polynomial time bounds as do the newer implementations. However, at this stage we have not spent significant resources to achieve the fastest possible implementations, and still hold out the hope of combining heuristic speed with a theoretical proof of correctness.

3.3. Elementary Near Optimal Group Membership.

This section provides a general framework for a randomized group membership algorithm. The algorithm is presented in greater detail, since it has not yet appeared in the literature. The underlying principles of this algorithm are all rather elementary in comparison to the algorithm described in [**5**] yet we are still able to achieve a reduction in the time complexity from $O(n^3 \log^4 n)$ to $O(n^2 \log |G| \log n)$ with reliability at least $1 - 1/n$.

In effect, the algorithm of this section provides a theoretically sound mechanism for a fast Sims-type group membership using random elements. It has been observed by several authors, both theoretically [**2**] and heuristically in implementations [**29**], that if one could efficiently construct random elements of an arbitrary group, G, from generators of G, then one could derive a faster group membership algorithm. Typically, such an algorithm finds random generators of $G^{(2)}$ according to some distribution, and then repeats the process. This section continues that tradition, but also invokes the idea of cubes [**10**], as specialized to Schreier trees [**6**] (see §2.7), and maintains several global data structures, $S^{(i)}$, $C^{(i)}$, and $D^{(i)}$.

The global data structure $S^{(i)}$ represents a set of group elements of $G^{(i)}$ to be constructed, such that $\cup_{j=i}^{n-1} S^{(j)}$ generates $G^{(i)}$. $C^{(i)}$ is the cube over $S^{(i)}$. (For $S^{(i)} = \{g_{i,1}, \ldots, g_{i,k}\}$, $C^{(i)} = g_{i,1}^{e_1} \cdots g_{i,k}^{e_k}$ for $e_j \in \{0, 1\}$.) The cube $D^{(i)} = C^{(n-1)} C^{(n-2)} \cdots C^{(i)}$ is derived from $C^{(i)}$, and is used purely as a notational convenience to describe the algorithm. Initially, $S^{(i)} = \emptyset$ and $D^{(i)} = C^{(i)} = \{1\}$. One need not store any data structure corresponding to $D^{(i)}$. The points $\{\alpha_i\}$ are the permutation domain. Sets such as $\alpha_i^{C^{(i)}}$ and $\alpha_i^{D^{(i)^{-1}} D^{(i)}}$ are re-computed whenever needed (although caching schemes are possible).

The algorithm is presented in a top-down manner. The overall control structure is that of a Sims-type algorithm: development of the point stabilizer sequence in a top-down fashion. The auxiliary routine *Random-Point-Stabilizer-Element* returns randomized elements of the point stabilizer subgroup, that play the role of Schreier generators. The auxiliary procedure *Deep-Sift* is used for "sifting" or "stripping" these pseudo-Schreier generators. Calculation of the pseudo-Schreier generators requires prior calculation of a transversal for the point stabilizer subgroup. This is the job of *Complete-Fundamental-Orbit*.

The algorithm is described in a somewhat general setting because we believe that the control structure can be modified to work with more specific classes of groups, such as small base groups, by invoking a different version of the auxiliary routine *Random-Point-Stabilizer-Element*. This routine should have the property that while $\langle D^{(i+1)} \rangle \neq G^{(i+1)}$, *Random-Point-Stabilizer-Element(i)* returns an element of $G^{(i+1)} \setminus \langle D^{(i+1)} \rangle$ with probability at least p. The implementation of *Random-Point-Stabilizer-Element* returns a random Schreier subproduct which works in all cases with $p = 1/4$ by Lemma 2.2. If a different version is used, then it may require a different value for p. Additionally, the two constants c and c' must be chosen based on the differing probability p.

Procedure *Strong-Generating-Set*(S, r)
Input: generating set $S \subseteq G$ and $r \geq n$
Parameters: constants $c > 0$, $0 < c' < 1$ chosen appropriately
Output: strong generating set $\{S^{(i)}\}$
Time: $O(nb \log |G| \log r + k' b \log r + n|S| \log |G|)$,
where k' is the time for a single call to *Random-Point-Stabilizer-Element*

Reliability: at least $1 - 1/r$, provided $r \geq n$
    For $i \leftarrow 1$ to $n-1$ do
        Initialize $S^{(i)} \leftarrow \emptyset$
        Initialize $C^{(i)} \leftarrow \{1\}$
    Set $S^{(1)} \leftarrow S$
    For $i \leftarrow 1$ to $n-1$ do
        If $S^{(i)} \neq \emptyset$ then [$S^{(i)}$ constructed during previous calls to *Deep-Sift*]
            Complete-Fundamental-Orbit(i)
            [We now have generators for $G^{(i)}$
              and coset representatives for $G^{(i)}/G^{(i+1)}$]
            Repeat
                Initialize $successes \leftarrow failures \leftarrow 0$
                Loop for $c \log r$ iterations [c appropriate constant]
                    Set $g \leftarrow$ *Random-Point-Stabilizer-Element*($i+1$)
                    [While $\langle D^{(i+1)} \rangle \neq G^{(i+1)}$, Prob($g \in G^{(i+1)} \setminus \langle D^{(i+1)} \rangle$) $\geq p$.]
                    Deep-Sift(i+1, g)
                    If *Deep-Sift* returned "success" then
                        $successes \leftarrow successes + 1$
                    Else $failures \leftarrow failures + 1$
            Until $successes < c'(successes + failures)$ [for a constant $c' < p$]
            [Now, $\langle D^{(i+1)} \rangle = G^{(i+1)}$]
    Return($\{S^{(i)}\}$)

Procedure *Deep-Sift*(i, g)
Input: $1 \leq i \leq n-1$; $g \in G^{(i)}$
Output: Either a level $i' \geq i$ and $g' \in G^{(i')}$ such that $|\alpha_{i'}^{C^{(i')}\{g',1\}}| = 2|\alpha_{i'}^{C^{(i')}}|$ or else "fail"; "fail" occurs only if $g \in {D^{(i)}}^{-1} D^{(i)}$
Time: $O(n \log |G|)$
    If $|\alpha_i^{C^{(i)}\{g,1\}}| = 2|\alpha_i^{C^{(i)}}|$ then [$g \notin {C^{(i)}}^{-1} C^{(i)}$]
        Set $S^{(i)} \leftarrow S^{(i)} \cup \{g\}$
        Set $C^{(i)} \leftarrow C^{(i)}\{g, 1\}$
        Return(i, g)
    Else if $i = n-1$ then
        Return("fail")
    Else [$\alpha_i^{C^{(i)}g} \cap \alpha_i^{C^{(i)}} \neq \emptyset$]
        Set $u \leftarrow$ arbitrary element of $C^{(i)}$ such that ${\alpha_i}^u \in {\alpha_i}^{C^{(i)}g^{-1}}$
        Set $u' \leftarrow$ arbitrary element of $C^{(i)}$ such that ${\alpha_i}^{ug} = {\alpha_i}^{u'}$
        Return(*Deep-Sift*(i+1, ugu'^{-1}))

Verifying the output property of *Deep-Sift* reduces to demonstrating that the property $g \notin {D^{(i)}}^{-1} D^{(i)}$ is preserved in recursive calls. To see this, note that $g \notin {D^{(i)}}^{-1} D^{(i)} \Rightarrow D^{(i)} g \cap D^{(i)} = \emptyset \Rightarrow D^{(i+1)} ug \cap D^{(i+1)} u' = \emptyset \Rightarrow D^{(i+1)} ugu'^{-1} \cap D^{(i+1)} = \emptyset \Rightarrow ugu'^{-1} \notin {D^{(i+1)}}^{-1} D^{(i+1)}$. Let j be the smallest k such that

$D^{(k)} = C^{(k)}$. If $g \notin {D^{(i)}}^{-1} D^{(i)}$ then *Deep-Sift* returns (i', g') for which $i \leq i' \leq j$ and $g' \in G^{(i')} \setminus {C^{(i')}}^{-1} C^{(i')}$. This satisfies the output property.

Further, it will be crucial in later arguments to observe that no matter how often *Deep-Sift* is successively called with different arguments, there can never be more than $\log |G|$ successful calls to *Deep-Sift*. This follows from observing that the number of elements added to the cube $C^{(i)}$ can never grow to more than $\lfloor \log n_i \rfloor$ for each i. Further, the bound $\log n_i$ on the length of the cube $C^{(i)}$ shows that $O(n \log n_i)$ time is spent at the i^{th} level, and $O(n \log |G|)$ time can be spent over all recursive calls from *Deep-Sift*.

Procedure *Complete-Fundamental-Orbit*(i)
Input: $1 \leq i \leq n-1$
Output: $\{C^{(i)}\}$ such that $\alpha_i^{{D^{(i)}}^{-1} D^{(i)}} = \alpha_i^{\langle D^{(i)} \rangle}$
Time: $O(n \log |G| + k)$, where k is the time for all subsequent calls to Deep-Sift
  While $\alpha_i^{{D^{(i)}}^{-1} D^{(i)}} \neq \alpha_i^{\langle D^{(i)} \rangle}$ do
    [Note that $\langle D^{(i)} \rangle = \langle \cup_{j \geq 1} S^{(j)} \rangle$]
    Let $u \in \langle D^{(i)} \rangle$ be such that ${\alpha_i}^u \notin \alpha_i^{{D^{(i)}}^{-1} D^{(i)}}$
    Deep-Sift(i, u)
  Return$(\{C^{(i)}\})$

Since the length of $D^{(i)}$ will always be bounded by $\log |G|$, computation of images of α_i can be done in $O(n \log |G|)$ time. At most one such set of computations occurs without a corresponding call to *Deep-Sift*, and the remaining computations are clearly dominated by time for the corresponding call to *Deep-Sift*, $O(n \log |G|)$. Hence the overall time of $O(n \log |G| + k)$ follows.

THEOREM 3.6. *Assume that Random-Point-Stabilizer-Element*$(i+1)$ *(as described in the code for Strong-Generating-Set) produces a g satisfying* $\text{Prob}(g \in G^{(i+1)} \setminus \langle D^{(i+1)} \rangle) \geq p$ *(for fixed p with $0 < p < 1$) whenever $\langle D^{(i+1)} \rangle \neq G^{(i+1)}$. If k' is the time for a single call to Random-Point-Stabilizer-Element, then Strong-Generating-Set completes in time $O(nb \log |G| \log r + k' b \log r + n|S| \log |G|)$ with reliability $O(1 - 1/r)$ for $r \geq n$ for appropriate values of the constants c and c'.*

PROOF. Note that *Strong-Generating-Set* makes at most $((1/c') \log |G| + cb \log r)$ calls to *Deep-Sift*. This is seen by noting that the "Until" condition of the the "Repeat ... Until" block will not trigger while $successes \geq c'(successes + failures)$. Thus, this condition will hold on all except the last iteration of the block. On these earlier iterations, $successes + failures \leq (1/c') successes \leq (1/c') \log |G|$ over the life of the algorithm. This follows since *successes* can be identified with the number of successful calls to *Deep-Sift*, which was previously shown to be bounded by $\log |G|$. Beyond that, when $successes < c'(successes + failures)$, there are exactly b iterations of the "Repeat ... Until" block with $c \log r$ calls to the inner "Loop". Thus, there are at most $((1/c') \log |G| + cb \log r)$ calls to *Deep-Sift*.

The time for each call to *Random-Point-Stabilizer-Element* and *Deep-Sift* is k' and $n \log |G|$, respectively. From the previous paragraph, we know that they are each called at most $((1/c') \log |G| + cb \log r)$ times. So, over the life of the algorithm, the calls to those routines from *Strong-Generating-Set* account for at most $((1/c') \log |G| + cb \log r)\,(n \log |G| + k') = O(nb \log |G| \log r + k'b \log r)$ time when $r \geq n$.

Next, we must bound the time for calls to *Complete-Fundamental-Orbit*. Since there are at most $O(\log |G| + b \log r)$ calls to *Deep-Sift* over the life of the algorithm, the time bound is $O(nb \log |G| + nb \log |G| \log r + k'b \log r)$, which is dominated by the previous times for $r \geq n$.

Finally, we prove the stated reliability. At level i, as long as $G^{(i+1)} \neq \langle D^{(i+1)} \rangle$ throughout the execution of the "Repeat... Until" block, we would like the probability that $successes \geq c'(successes + failures)$ to be sufficiently high. This will ensure that the termination criteria will not be invoked prematurely and the algorithm will not proceed incorrectly to level $i + 1$. By the above analysis, there are at most $(1/c') \log |G|/(c \log r) = \log |G|/(cc' \log r)$ blocks for which $successes \geq c'(successes + failures)$. In order to have at least $1 - 1/r$ overall reliability, the probability of exiting a single block with the termination criterion $successes < c'(successes + failures)$ while $G^{(i+1)} \neq \langle D^{(i+1)} \rangle$ should be at most $1/(r \log |G|/(cc' \log r))$.

We invoke Chernoff's bound (see §2.1) with $t = successes + failures = c \log r$ and $S_t = successes$. Set $c' = (1 - \epsilon)p$. (The parameter p is determined by the particular procedure *Random-Point-Stabilizer-Element*.) To ensure an overall probability of error of at most $1/r$ among all $\log |G|/(cc' \log r)$ blocks, or an error of at most $1/(r \log |G|/(cc' \log r))$ for each block, we invoke Chernoff's bound to satisfy

$$\begin{aligned} \text{Prob}(S_t < c't) &\leq \text{Prob}(S_t \leq \lfloor (1 - \epsilon)pt \rfloor) \\ &\leq e^{-\epsilon^2 pc(\log r)/2} \leq \frac{1}{r} \cdot \frac{1}{\log |G|/(cc' \log r)}, \qquad \text{for } c' = (1 - \epsilon)p. \end{aligned}$$

If $r \geq n$ (which implies $r \geq b \geq \log |G|/\log r$), this will be true if $\epsilon^2 pc(\log r)/2 \geq \ln(r^2/(cc'))$ for some value of c, since $\ln(r^2/(cc')) \geq \ln(r \log |G|/(cc' \log r))$. The inequalities $r \log r \geq \log |G|$ and $|G| \geq 2$ imply that $r > 1.5$, and it is clear that some value of c will satisfy the previous inequality for $r > 1.5$. This will then yield the reliability of $1 - 1/r$. □

COROLLARY 3.7. *If the procedure Random-Point-Stabilizer-Element returns a random Schreier subproduct, then for $r = n \geq 10$ and parameter values $c' = 0.05$ and $c = 20$, the algorithm Strong-Generating-Set performs in $O(n^2 \log |G| \log n + n|S| \log |G|)$ time with reliability at least $1 - 1/n$.*

PROOF. Note that a random Schreier subproduct has probability at least $p = 1/4$ of lying in $G^{(i+1)} \setminus \langle D^{(i+1)} \rangle$. The time $O(n^2)$ for a random Schreier subproduct will enter into the time $k' = O(n \log |G| + n^2)$ for *Deep-Sift*. However,

the term $k'b \log r$ will still be bounded by $n^2 \log |G| \log n$. The rest is easily seen by setting $\epsilon = 0.8$, noting that $c' = (1-\epsilon)p$, and verifying the Chernoff inequality of the previous proof for $c = 20$. $\square$

4. Base Change

Let G be a subgroup of S_n and let S be a strong generating set for the point stabilizer sequence of G relative to an ordering $\alpha = \alpha_1, \ldots, \alpha_n$ of $\{1, 2, \ldots, n\}$. A *change of base* is the construction of a strong generating set S' for G relative to a new ordering α'. Base change is a crucial algorithm in many important group computations and plays an especially important role in many of the backtracking algorithms currently used to solve problems for which efficient algorithms are not currently known. A *cyclic base change* occurs when α' is obtained from α through a *right cyclic shift*. In this case, α and α' satisfy the relationship

$$\alpha = \alpha_1, \ldots, \alpha_{r-1}, \alpha_r \ldots, \alpha_{s-1}, \alpha_s, \alpha_{s+1}, \ldots, \alpha_n$$
$$\alpha' = \alpha_1, \ldots, \alpha_{r-1}, \alpha_s, \alpha_r, \ldots, \alpha_{s-1}, \alpha_{s+1}, \ldots, \alpha_n$$

It is this special case of base change that is usually required in many backtracking algorithms [**12, 13, 30**].

We quickly review existing algorithms for performing a change of base and then describe new deterministic and randomized results for the special case of cyclic base change.

The first algorithm for performing a change of base was given by Sims [**35**]. This algorithm employs a clever trick and uses a Schreier vector data structure for representing the point stabilizer sequence. Sims's trick was generalized in [**11**] to show that a deterministic cyclic base change could be performed in time $O(n^2)$ using $O(n^2)$ space. This improved the running time of Sims's algorithm from $O(n^5)$ to $O(n^3)$. This speedup was due in part to the use of a new data structure for representing the point stabilizer sequence, first described by Jerrum [**28**] and referred to as a labeled branching,

The labeled branching data structure requires $\Theta(n^2)$ storage for transitive G and so is impractical when n is large. In the context of a base change, one already has a strong generating set S for G relative to α and a Schreier vector data structure $\{\mathcal{T}_i\}_{i=1}^n$ which can be used to compute random elements of G. Ideally, one would like to use $\{\mathcal{T}_i\}_{i=1}^n$ to compute random elements in time $O(n \log |G|)$. If the sum of the depths of $\mathcal{T}_i$ is $O(\log |G|)$, then this is the case and $\{\mathcal{T}_i\}_{i=1}^n$ is said to be a *short Schreier vector* data structure. Under this assumption, the randomized base change algorithm described in [**18**] takes time $O(n \log^2(|G|))$ and builds a short Schreier vector data structure relative to α'. The following result ensures that with some additional preprocessing time, one can always assume that $\{\mathcal{T}_i\}_{i=1}^n$ is short.

PROPOSITION 4.1. (from [**19, Theorem C**]) *Given a strong generating set S for G relative to α, one can compute with reliability $1 - 1/n$ a short Schreier vector data structure in time $O(n \log^2 |G| + n|S| + n \log n)$.*

The proof of Proposition 4.1 uses the methods of §3.1.1 and §3.1.2 in a novel way to achieve the time bound. For insight into how this is accomplished, assume that $\{\mathcal{T}_j\}_{j=i+1}^{n}$ is a short Schreier vector data structure for $G^{(i+1)}$ (relative to α) which we want to extend to $G^{(i)}$. Let $n_i = |{\alpha_i}^{G^{(i)}}|$. The first step is to build a sequence R of elements of $G^{(i)}$ such that $|G^{(i+1)}C(R)| = |G^{(i+1)}|2^{|R|}$. If R is a maximal length sequence with this property, then $|R| \leq \log n_i$ and

$$ {\alpha_i}^{G^{(i)}} = {\alpha_i}^{C(R)^{-1}G^{(i+1)}C(R)}. $$

Since each element of $G^{(i+1)}$ can be written as a word in the labels of $\mathcal{T}_j$, $j < i$, of length at most $c\log(|G^{(i+1)}|)$, it then follows that each coset representative of $G^{(i)}$ can be computed as a word of length at most $c\log|G^{(i)}|$. In particular, a random element for $G^{(i)}$ can be computed using at most $c\log|G^{(i)}|$ multiplies. We can now use Proposition 3.1 to build a short Schreier tree $\mathcal{T}_i$ for ${\alpha_i}^{G^{(i)}}$ using at most $c\log n_i$ random elements. The timing for the i^{th} iteration can be shown to be $O(n\log|G^{(i)}|\log n_i)$ (assuming that $|S| \leq \log|G|$) and this leads to the final result.

THEOREM 4.2. (from [**18, Theorem 5.3**]) *Given a strong generating set S for G relative to an ordering α, one can with reliability $1 - 1/n$ perform a base change relative to an arbitrary ordering α' in $O(n\log^2|G| + n|S| + n\log n)$ time using $O(n\log|G|)$ space. Furthermore the algorithm returns a short Schreier vector data structure with respect to the new ordering.*

The next series of results concerns the important case of a cyclic base change. A *cube Schreier vector data structure* is one for which each Schreier tree can be represented by a cube $C^{(i)}$ of length at most $\log|G^{(i)}|$ such that $\alpha_i^{G^{(i)}} = \alpha_i^{{C^{(i)}}^{-1}C^{(i)}}$.

THEOREM 4.3. (from [**18, Theorem A**]) *Given a strong generating set S for G, a deterministic cyclic base change algorithm can be described that requires $O(n\log^2|G| + n|S| + n\log n)$ time and $O(n\log|G|)$ space. Furthermore, the algorithm returns a cube Schreier vector data structure for G relative to the new ordering.*

The key to the proof of Theorem 4.3 is the technique described in [**11**] in conjunction with the following result on the construction of cube Schreier trees.

PROPOSITION 4.4. (from [**19, Proposition 2.3**]) *Let S be a strong generating set for G relative to some ordering α. Then one can compute, in time $O(n\log^2|G| + n|S| + n\log n)$, both a sequence R of group elements of G such that $C(R)$ is non-degenerate and a complete cube Schreier vector data structure $\{\mathcal{T}_i\}_{i=1}^{n}$ with the following property. For each $\mathcal{T}_i$, $\texttt{Labels}(\mathcal{T}_i) \subseteq R_i \cup R_i^{-1}$ where $R_i = G^{(i)} \cap R$ is a prefix of R of length at most $\log|G^{(i)}|$, for $1 \leq i \leq n$. In particular, $\{\mathcal{T}_i\}_{i=1}^{n}$ requires $O(n\log|G|)$ space.*

The proof of Proposition 4.4 follows easily from the ideas used in the procedure *Build-Cube-Schreier-Tree*. It is interesting to note that within the same time

bound, one can build a sequence R with $C(R)$ non-degenerate, such that every orbit of $G^{(i)}$, $1 \leq i \leq n$, can be represented by a cube Schreier tree with labels chosen from a suitable prefix of R.

Using randomized methods one can give a substantial theoretical improvement to Theorem 4.3 under the hypothesis of availability of fast generation of random group elements. This is satisfied if a *short Schreier vector data structure* is available.

THEOREM 4.5. (from [**19, Theorem B**]) *Assume random elements can be computed in time $O(n \log |G|)$. Let b be the size of a base relative to some ordering α and let α' be an ordering obtained from α by a right cyclic shift. Then a randomized cyclic base change algorithm can be described which has probability at least $1 - 2/n$ of using $O(nb \log^2 n)$ time and $O(nb \log n)$ space. Furthermore the algorithm returns a short Schreier vector data structure with respect to the new ordering.*

The proof of Proposition 4.5 depends on the following observation: If H is a subgroup of a finite group G, U a complete set of right coset representatives for H in G and $S \subseteq G$ a set of mutually independent uniformly random elements of G, then $T = \{g\bar{g}^{-1} : g \in S\}$ is a set of mutually independent and uniformly random elements of H. (For $g \in G$, $\bar{g}$ is the unique element of U so that $H\bar{g} = Hg$.) This observation is applied in the context of sifting down a subgroup chain of a permutation group G. The subgroup chain is chosen consistently with the problem of performing a right cyclic shift. Initially, we have a set of $c \log n$ mutually independent elements of G for some constant c. As we move down the subgroup chain, each element is multiplied by a suitable coset representative to produce a set of $c \log n$ mutually independent random elements for the next subgroup in the chain. These random elements are sufficient to build a short Schreier tree for that next subgroup using Proposition 3.1.

5. Reduction of Other Constructions to Group Membership

Another important construction [**20**] shows how to reduce certain group constructions to the point stabilizer construction (equivalent to group membership). In some situations, a partial strong generating set will already be known, and efficiency will be further improved through a base change algorithm. By carefully choosing generators of an initial group, and constructing a point stabilizer sequence with respect to a specified ordering of the points, one can then change the ordering and extract other groups. This method turns out to be surprisingly powerful, yielding efficient algorithms for normal closure, center, commutator subgroups and intersection of two groups, when one group is contained in the normalizer of the other. Further, all constructions yield a strong generating set for the constructed group in essentially the time to carry out the point stabilizer sequence, or less.

Let G and H be distinct permutation groups acting on an n-element set A. Let

$\tilde{A} = A_1 \dot{\cup} A_2$, where A_1 and A_2 are copies of A. Let D be the subgroup of $Sym(\tilde{A})$ consisting of all elements of the form $\{(g,g) : g \in G\}$. Let $K = \langle D \cup (1 \times H) \rangle \subseteq G \times G$. Since each element of K can be written uniquely in the form (g_1, g_2) for suitably chosen elements $g_1 \in G$, and $g_2 \in \langle H, G \rangle$, we can define projection maps $f_1 : K \to G$, and $f_2 : K \to \langle H, G \rangle$ according to the rules $f_1((g_1,g_2)) = g_1$ and $f_2((g_1,g_2)) = g_2$. Finally, let $K_1 = f_1(K_{A_2})$ and $K_2 = f_2(K_{A_1})$ where K_{A_1} and K_{A_2} are the pointwise stabilizer subgroups in K of A_1 and A_2 respectively. To fix notation, for elements x, y of a group we refer to $x^{-1}yx$ as the *conjugate* of y by x and denote it by y^x.

THEOREM 5.1. (from [**20, Theorem 2.1**]) *Under the above conditions,* $K_1 = \langle H^G \rangle \cap G$ *and* $K_2 = \langle H^G \rangle$.

PROOF. Observe that for sufficiently large k, an arbitrary element of K can always be expressed in the form $\bar{g}_0 \bar{h}_1 \bar{g}_1 \bar{h}_2 \bar{g}_2 \cdots \bar{h}_k \bar{g}_k$, where $\bar{g}_1, \cdots, \bar{g}_k \in D$ and $\bar{h}_1, \cdots, \bar{h}_k \in 1 \times H$. Letting $\bar{g}_i = (g_i, g_i)$ and $\bar{h}_i = (1, h_i)$ (1 is the identity), it follows that

$$\begin{aligned} K &= \{(g_0 g_1 \cdots g_k,\, g_0 h_1 g_1 h_2 g_2 \cdots h_k g_k) \mid g_i \in G, h_i \in H\} \\ &= \{(g_0 g_1 \cdots g_k,\, g_0 g_1 \cdots g_k h_1^{g_1 \cdots g_k} h_2^{g_2 \cdots g_k} \cdots h_k^{g_k}) \mid g_i \in G, h_i \in H\}. \end{aligned}$$

The last equation is similar to the one used to prove Schreier's lemma [**26, Lemma 7.2.2**]. In light of this description of K, we can "solve for g_0" in order to find the following characterizations of K_1 and K_2.

$$\begin{aligned} K_1 &= \{g_0 g_1 \cdots g_k \mid g_0 g_1 \cdots g_k h_1^{g_1 \cdots g_k} h_2^{g_2 \cdots g_k} \cdots h_k^{g_k} = 1, g_i \in G, h_i \in H\} \\ &= \{(h_1^{g_1 \cdots g_k} h_2^{g_2 \cdots g_k} \cdots h_k^{g_k})^{-1} \mid (h_1^{g_1 \cdots g_k} h_2^{g_2 \cdots g_k} \cdots h_k^{g_k})^{-1} = g_0 g_1 \cdots g_k \in G, \\ &\qquad\qquad g_i \in G, h_i \in H\} \\ &= \langle H^G \rangle \cap G \end{aligned}$$

$$\begin{aligned} K_2 &= \{g_0 g_1 g_2 \cdots g_k h_1^{g_1 \cdots g_k} h_2^{g_2 \cdots g_k} \cdots h_k^{g_k} \mid g_0 g_1 \cdots g_k = 1, g_i \in G, h_i \in H\} \\ &= \{h_1^{g_1 \cdots g_k} h_2^{g_2 \cdots g_k} \cdots h_k^{g_k} \mid g_1, \ldots, g_k \in G, h_1, \ldots, h_k \in H \text{ are arbitrary}\} \\ &= \langle H^G \rangle \quad \square \end{aligned}$$

One of the most interesting deductions from this method is that construction of strong generators for a normal closure is no harder than construction of strong generators for a group of degree at most twice the original degree. Since group membership is in $O^{\sim}(n^3)$ time for large base groups, the time for normal closure is approximately a factor of 8 times slower than group membership. For small base groups, group membership is in $O^{\sim}(n)$ time, and the time will be approximately a factor of 2 slower.

The utility of the theory has been enhanced by our recent results. Consider the situation where G normalizes H. Under this assumption, it follows that K can be factored as $K = D(1 \times H)$. If in addition, strong generating sets are known for G and H respectively, then this factorization can be used to directly

compute a strong generating set for K [**20, Lemma 2.2**]. This reduces the computation of $G \cap H$ to that of a base change. Strong generating sets for G and H can be computed in time $O(n^2 \log(|G||H|) \log n + n(|S_G| + |S_H|) \log(|G||H|))$ (Corollary 3.7) with reliability at least $1-1/n$. Note that $|K| \leq |G||H|$. Applying Theorem 4.2 allows one to perform a randomized base change on K in time $O(n \log^2(|G||H|) + n(|S_G| + |S_H|) + n \log n)$, which then yields the following enhancement of [**20, Theorem 1(iii)**].

THEOREM 5.2. *Suppose G and H act on n points and G normalizes H. Let G and H be generated by S_G and S_H respectively. Then a strong generating set for $G \cap H$ can be computed in time $O(n^2 \log(|G||H|) \log n + n(|S_G| + |S_H|) \log(|G||H|))$ with reliability at least $1 - 2/n$.*

In a journal version of [**20**], the authors together with Luks will in fact extend the method described in Theorem 5.1 to show that additional group constructions may be reduced to the point stabilizer sequence. One that is particularly interesting and indicates the flavor of the method is the construction of the commutator subgroup. Using the notation earlier, let $G \subseteq Sym(A)$ and and let $\tilde{A} = A_1 \dot{\cup} A_2 \dot{\cup} A_3$, where A_1, A_2 and A_3 are copies of A. Let $D_1 \subseteq Sym(A_1 \dot{\cup} A_2)$ consist of all elements of the form $\{(g, g) : g \in G\}$ and let D_2 be the analogous subgroup of $Sym(A_2 \dot{\cup} A_3)$. Let $K = \langle (D_1 \times 1) \cup (1 \times D_2) \rangle$. Then the projection K_3 of $K_{A_1 \dot{\cup} A_2}$ onto A_3 is the commutator subgroup of G.

6. Compact Data Structures for Cayley and Schreier Coset Graphs

Given a generating set Φ for a group G, the *minimum word problem* is to express an arbitrary element of G as a word in Φ of minimal length in $\Phi \cup \Phi^{-1}$. This problem is known to be *P-space complete* [**27**]. Nevertheless, one practical solution is to calculate a minimum depth spanning tree of the Cayley graph for the group relative to Φ. The *Cayley graph* of G with respect to generators Φ is a labeled graph $\widetilde{G}$ such that the nodes of $\widetilde{G}$ correspond to the group elements of G, and an edge (g, h) of $\widetilde{G}$ (for $g, h \in G$) is labeled by a generator $\phi \in \Phi \cup \Phi^{-1}$ such that $g\phi = h$.

We have developed two distinct techniques [**16**] to construct spanning trees. One is more space-efficient. It requires $\log_2 5$ bits per node ($|G|(\log_2 5)/8$ bytes) plus a much smaller amount of additional space for temporary data structures. However, as part of a space-time tradeoff, it operates in time $O(|G|(d + |\Phi|b^2))$ instead of the $O(|G||\Phi|)$ time of a traditional breadth-first implementation. (Here b is the base size of G represented as a permutation group of degree n, and d is the diameter of the Cayley graph.) The second technique is more CPU-efficient than the traditional breadth-first implementation, while retaining comparable space usage. While we have analyzed the formal worst-case complexity in terms of some more obscure parameters, the *typically* observed time in computer experiments is more relevant. It was $\Theta(|G|(b + d))$. Computer experiments have convinced us that for most computers and groups, space is more of a bottleneck than time,

and so the first technique is the more valuable one.

Because the space-efficient data structure requires $\log_2 5$ bits per node of intermediate storage and $\log_2 3$ bits of final storage, we have referred to this as a 2-bit data structure, where $\log_2 4 = 2$ bits was jokingly considered as an average of $\log_2 3$ and $\log_2 5$ bits. An alternative formulation exists that uses only 2 bits per node for intermediate storage, but at the cost of some additional time.

The 2-bit data structure was used [**23**] to enumerate the group of order 3,674,160 for Rubik's 2-cube using only 1 megabyte of RAM on a SUN-3. Where $|G|$ is still larger, even the space-efficient method runs out of space, and approximate solutions (short words) need to be obtained by using a chain of subgroups and developing spanning trees for the respective *Schreier coset graphs* [**16**]. Finally, the techniques can often be generalized to other abstract groups containing an effective group multiplication and an effectively computable dense mapping between the group elements and the integers.

The 2-bit method can best be understood by comparison with a standard breadth-first search algorithm for building a spanning tree for a Cayley graph. Such an algorithm would require an array of length $|G|$. It might use hashing into an array to determine if a node labeled by a permutation has previously been seen. Pointers to neighbors, or at least to the parent, could then be stored in each entry.

There are two important principles for designing the 2-bit data structure. First, we assume a perfect, invertible hashing function, f. This is an encoding function $f : G \to \{0, 1, \ldots, |G| - 1\}$ along with its inverse. It is well known in computational group theory [**16**] that such a function is efficiently computable. This provides an implicit index into the array, so that the key (the permutation) need not be stored for each array slot.

Second, each array entry need not store pointers to some of the adjacent nodes. Instead, it suffices to store one of five values in each entry. (Hence, the requirement for $\log_2 5$ bits per node.) The numbers 0 to 2 indicate that the distance is known from the current node to the root modulo 3. (This is equivalent to the depth, if we view the data structure as representing a tree.) The number 3 represents a (frontier) node which is adjacent to a node with label from 0 to 2. The number 4 is for all other (unexplored) nodes.

Note that this construction suffices to find a shortest path from a node g to h. When considering the path as a sequence of labels drawn from the generating set Φ, this is the same as a path from the node $h^{-1}g$ to the identity. So, the problem reduces to finding a shortest path from a node to the root. Shortest paths are determined by repeatedly determining parent nodes until the root node is reached. Formally, the process reduces to calling *parent*(), initially on $f(h^{-1}g)$, and then calling it repeatedly on the result until the root node is reached. The function *parent* and associated data structures are defined as follows.

Let D be an array with $|G|$ entries (indexed from 0 to $|G| - 1$) capable of

storing 5 distinct values. For $i \in [0, |G|-1]$, let

$$\textit{node-neighbor}(i) = \{j \in [0, |G|-1]:\ f^{-1}(i)\, g = f^{-1}(j) \text{ for some } g \in \Phi\}.$$

For i such that $D[i] \in \{0,1,2\}$, define $\textit{parent}(i) = \{j \in \textit{node-neighbor}(i):\ D[j] = D[i]-1 \mod 3\}$. Note that for all nodes except the root node, $\textit{parent}(i)$ must be non-empty, since $D[i]$ can be interpreted as the depth of a spanning lattice. $\textit{parent}(i)$ may have more than one element, and so the data structure should be interpreted as a spanning lattice rather than a spanning tree.

Procedure *Build-Compact-Data*
Input: A generating set Φ *for* G
Output: The array D giving the 2-bit data structure.
Global Variables: A labeled branching data structure for G relative to α (or any other data structure, such as a Schreier vector, for computing f and f^{-1} in *node-neighbor*).

```
Initialize D[0] ← 0 and all other entries in D ← 4 [unexplored]
For each k ∈ node-neighbor(0), set D[k] ← 3 [frontier]
Set level ← 0
[Loop through nodes which are at the current level.]
LOOP:
    Set level ← level + 1
    For i ← 0 to |G| − 1 do
        If D[i] = 3 then
            If ∃j ∈ node-neighbor(i) with D[j] = level − 1 mod 3 then
                Set D[i] ← level mod 3
                For each k ∈ node-neighbor(i) with D[k] = 4 do
                    Set D[k] ← 3 [frontier]
    If any new nodes were set to 3 in current loop, go to LOOP
    Else return D
```

THEOREM 6.1. (from [**16, Theorem 4.3**]) *Let T be the time for each call to the encoding function f, and T' the time for each call to f^{-1}. Then the array D can be constructed in time $O(|G|(d + |\Phi|T + T'))$ using space of $\log_2 5$ bits per node plus space and time required for building group membership data structures associated with f.*

The *Schreier coset graph* of G with respect to a subgroup H and generating set Φ is a labeled graph $\widetilde{G}$ such that the nodes of $\widetilde{G}$ correspond to the set of right cosets G/H of G, and an edge $(Hg, H\bar{g})$ of $\widetilde{G/H}$ (for $Hg, H\bar{g} \in G/H$) is labeled by a generator $\phi \in \Phi \cup \Phi^{-1}$ such that $Hg\phi = H\bar{g}$. The 2-bit data structure for storing Cayley graphs extends to Schreier coset graphs relative to a subgroup H as long as there is an efficiently computable $1-1$ function $f_H : G/H \to \{0, 1, \ldots, |G|/|H|-1\}$. In the case of permutation groups, we developed such a function [**16**]. Previously, Richardson had developed a technique [**31**] for finding lexicographically smallest elements of a left coset suitable for hashing of

cosets. This function f_H acts as a perfect hash function, and requires the newer cyclic base changes of §4 for efficiency. Since the inverse of f_H is not necessarily efficiently computable, in many interesting examples a better alternative is the use of an intermediate subgroup K with $H \subset K \subset G$ such that $|G/K|$ and $|K/H|$ are sufficiently small to support an encoding function based on factoring.

7. Software Implementations

Most of the tools and algorithms described here have been implemented in our LISP system, comprising approximately 25,000 lines, built up with limited manpower over about five years. Primarily, this manpower consisted of ourselves and our Ph.D. student, Namita Sarawagi, all working on a very part-time basis. The system was also used for small projects done by other students. LISP was chosen because the interactive environment coupled with an integrated debugger is convenient for rapid prototyping, while not sacrificing speed in compiled mode. In a small number of time-critical routines (primarily permutation multiply and permutation inverse), C routines are linked in to achieve close to optimal speed. A second important feature is a wide variety of data types, and flexible coercion between those data types. This has been especially important in experimenting with new data structures.

Having our own software system available was especially important in the beginning, in exploring the effect of using different low-level data structures. This facility has given us valuable insights into why certain data structures sometimes behave better than others in practice. Conversely, better theoretical understanding of algorithms has often led us to improved implementations.

Thus, we are often able to closely correlate our experimental times with theoretically derived asymptotic complexities. This has been especially true in the case of randomized algorithms, where the worst case examples tend to be "blurred" by the randomness, and the probabilistic worst case analysis tends to be close to the average case. For example, this was our experience with a randomized base change algorithm [**18**], where, after scaling by a constant factor, experimental times agreed with theoretically predicted times within a factor of 1.5. This comparison was carried out over 10 permutation groups with orders ranging from 10^7 through 10^{55}, degrees ranging from 48 to 106,080, and base sizes ranging from 4 to 52.

Appendix. Randomized Algorithms: Terminology

Although the terminology below and its motivation are widely understood in theoretical computer science, some of it is less widely known among those in computational group theory. Since it is difficult to find a standard exposition, this brief review is included.

A *randomized algorithm* is one which, for a fixed input, may require varying amounts of time and return different outputs upon different invocations. In the

absence of statements to the contrary, there is no guarantee that a randomized algorithm will return a correct answer.

DEFINITION M_1. *A randomized algorithm is a Monte Carlo algorithm if there is a function $f(n)$ such that for any probability $p < 1$, there is a reliability parameter t such that if the algorithm is allowed to run for time $tf(n)$ on an input whose size is parametrized by n, then it computes the correct answer for that input with reliability (probability of success) at least p. In such a situation, one says that the algorithm runs in $f(n)$ Monte Carlo time.*

In descriptions of a Monte Carlo algorithm, the reliability parameter, t, is sometimes omitted as an explicit input if the algorithm is to be run with a fixed reliability. A Monte Carlo algorithm is *Las Vegas* if it never returns an incorrect answer. Hence, a Las Vegas algorithm may only return a correct answer or "don't know". We say that an algorithm *runs in $O(g(n))$ Monte Carlo (Las Vegas) time*, if it is a Monte Carlo (Las Vegas) algorithm running in $f(n)$ time with $f(n) = O(g(n))$.

DEFINITION M_2. (*exponential reliability) A Monte Carlo algorithm has exponential reliability if there is a function $f(n)$ and a reliability parameter t such that if the algorithm is allowed to run for time $tf(n)$ on an input whose size is parametrized by n, then it computes the correct answer for that input with reliability (probability of success) at least $1 - e^{-n}$. In such a situation, one says that the algorithm runs in $f(n)$ Monte Carlo time with exponential reliability.*

The next theorem motivates the importance of exponential reliability over multiple invocations of an algorithm.

THEOREM A.1. *Suppose algorithm A runs in $O(f(n))$ Monte Carlo (Las Vegas) time with exponential reliability. If algorithm A' consists of invoking A for $g(n) \geq 3$ times on an input of size at most $n + \ln(g(n))$, then a variation of algorithm A' runs in $O(f(n+\ln(g(n)))\, g(n))$ Monte Carlo (Las Vegas) time with exponential reliability. If f and g are both polynomial functions, then the variation of algorithm A' runs in $O(f(n)g(n))$ polynomial time.*

Proof. Note that for a fixed input size n, an exponentially reliable algorithm can achieve higher reliability by simulating a larger input size (larger n). Hence, a variation of algorithm A' can simulate an input to A of size $n + \ln(g(n))$. The variation then runs in the stated time. The probability of error of each invocation of A is at most $e^{-(n+\ln(g(n)))}$, and so the overall probability of error is at most $g(n)e^{-(n+\ln(g(n)))} = e^{-n}$, as stated in the theorem. □

In many applications, the reliability does not increase exponentially with the input size, but it does increase exponentially with the amount of time allotted to it (the reliability parameter t). The next theorem addresses that. It assumes that each invocation receives an input of the same size, and is allowed to run for the same time. It is clear how to extend this to more general settings.

THEOREM A.2. *Let algorithm A be a Monte Carlo algorithm such that if it is allowed to run for $ctf(n)$ time for some constant c and some reliability parameter t, then its reliability will be exponential in t (at least $1-e^{-t}$). If algorithm A' consists of invoking A for $g(n) \geq 3$ times and allowing A to run for $2ct\ln(g(n))f(n)$ time per invocation, then algorithm A' runs in $O(tf(n)g(n)\ln(g(n)))$ Monte Carlo (Las Vegas) time with reliability exponential in t.*

Proof. Let $t = 2t'\ln(g(n))$ be the reliability parameter for A (for t' an arbitrary parameter). If algorithm A' is an invocation of algorithm A for $g(n) \geq \lceil e \rceil$ times with each invocation running for $ctf(n)$ time, and if algorithm A' is allowed to run for $g(n)ctf(n)$ time, then the probability of an error in A' over the $g(n)$ trials of A is at most $g(n)e^{-t} = g(n)/(g(n))^{2t'} \leq 1/(g(n))^{t'} \leq 1/e^{t'}$. So, A' has reliability at least $1 - e^{-t'}$, and A' is a $O(f(n)g(n)\ln(g(n)))$ Monte Carlo algorithm with reliability parameter t'. □

Note that a $O(tf(n))$ Monte Carlo algorithm that is exponentially reliable in the reliability parameter t can also be considered as a $O(nf(n))$ Monte Carlo algorithm that is exponentially reliable (in the input n). However, the previous theorem provides a finer estimate for such algorithms.

Acknowledgements

The authors would like to acknowledge many important discussions with László Babai, Gene Luks, Namita Sarawagi, and Ákos Seress. The authors would also like to acknowledge William Kantor for his careful reading and many helpful comments in earlier versions of this manuscript.

References

[1] D. Aldous, *On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing*, Prob. Eng. and Informational Sciences **1** (1987), 33–46.

[2] L. Babai, *Monte-Carlo algorithms in graph isomorphism testing*, Université de Montréal Tech. Rep. D.M.S. 79-10, Dep. Math. et Stat., 1979.

[3] ———, *On the length of chains of subgroups in the symmetric group*, Comm. in Algebra **14** (1986), 1729–1736.

[4] ———, *Bounded round interactive proofs in finite groups*, SIAM J. Discr. Math. (to appear).

[5] L. Babai, G. Cooperman, L. Finkelstein, E.M. Luks, and Á. Seress, *Fast Monte Carlo algorithms for permutation groups*, Proc. 23rd ACM STOC, 1991, pp. 90–100.

[6] L. Babai, G. Cooperman, L. Finkelstein, and Á. Seress, *Nearly linear time algorithms for permutation groups with a small base*, Proc. of the 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC '91), Bonn, 1991, pp. 200–209.

[7] L. Babai and P. Erdős, *Representation of group elements as short products*, Annals of Discrete Mathematics **12** (1982), 27–30.

[8] L. Babai, E. Luks, and Á. Seress, *Fast management of permutation groups*, Proc. 29th IEEE FOCS, 1988, pp. 272–282.

[9] L. Babai and M. Szegedy, *Local expansion in symmetrical graphs*, Combinatorics, Probability and Computing **1** (1992) (to appear).

[10] L. Babai and E. Szemerédi, On the complexity of matrix group problems I, Proc. 25th IEEE FOCS, 1984, pp. 229–240.

[11] C.A. Brown, L. Finkelstein, and P.W. Purdom, *A new base change algorithm for permutation groups*, SIAM J. Computing **18** (1989), 1037–1047.

[12] G. Butler, *Computing in permutation and matrix groups* II*: Backtrack Algorithm*, Math. Comp. **39** (1982), 671–680.

[13] G. Butler and C. Lam, *Isomorphism testing of combinatorial objects*, J. of Symbolic Computation **1** (1985), 363–381.

[14] P.J. Cameron, R. Solomon and A. Turull, *Chains of subgroups in symmetric groups*, J. of Algebra **127** (1989), 340–352.

[15] H. Chernoff, *A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Annals of Math. Statistics **23** (1952), 493–507.

[16] G. Cooperman and L. Finkelstein, *New methods for using Cayley graphs in interconnection networks*, Discrete Applied Mathematics **37/38** (1992), 95–118 (to appear).

[17] ———, *A strong generating test and short presentations for permutation groups*, J. Symbolic Computation **12** (1991), 475–497.

[18] ———, *A random base change algorithm for permutation groups*, J. Symbolic Computation (to appear).

[19] ———, *A fast cyclic base change for permutation groups*, Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 92), 1992.

[20] G. Cooperman, L. Finkelstein and E. Luks, *Reduction of group constructions to point stabilizers*, Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 89), ACM Press, 1989, pp. 351–356.

[21] G. Cooperman, L. Finkelstein and P.W. Purdom, *Fast group membership using a strong generating test for permutation groups*, Computers and Mathematics (E. Kaltofen and S.M. Watt, eds.), Springer-Verlag, 1989, pp. 27–36.

[22] G. Cooperman, L. Finkelstein and N. Sarawagi, *A random base change algorithm for permutation groups*, Proc. of 1990 International Symposium on Symbolic and Algebraic Computation, (ISSAC 90), ACM Press and Addison-Wesley, 1990, pp. 161–168.

[23] ———, *Applications of Cayley graphs*, Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC-8) (S. Sakata, ed.), Lecture Notes in Computer Science **508**, Springer-Verlag, 1991, pp. 367–378.

[24] G. Cooperman, L. Finkelstein and B. York, *Parallel implementations of group membership and the method of random subproducts*, Proc. of 1992 Dartmouth Institute for Advanced Graduate Studies in Parallel Computation Symposium (DAGS'92) (also Northeastern University Technical Report NU-CCS-90-17) (D. Johnson, F. Makedon, and P. Metaxas, eds.), 1992, pp. 94–100.

[25] P. Erdős and A. Rényi, *Probabilistic methods in group theory*, J. d'Analyse Math. **14** (1965), 127–138.

[26] M. Hall, Jr., *The Theory of Groups*, Macmillan, 1959.

[27] M. Jerrum, *The complexity of finding minimal length generator sequences*, Theoretical Computer Science **36** (1985), 265–289.

[28] ———, *A compact representation for permutation groups*, J. Algorithms **7** (1986), 60–78.

[29] J. Leon, *On an algorithm for finding a base and strong generating set for a group given by a set of generating permutations*, Math. Comp **35** (1980), 941–974.

[30] ———, *Computing automorphism groups of combinatorial objects*, Computational Group Theory (M. D. Atkinson, ed.), Academic Press, 1984, pp. 321–337.

[31] D. Richardson, *GROUP: A Computer System for Group-theoretical Calculations*, M.Sc. Thesis, Univerity of Sydney, 1973.

[32] N. Sarawagi, G. Cooperman and L. Finkelstein, *Group membership for groups with primitive orbits*, this volume.

[33] ———, *Computational Group Theory and Applications to Search*, Ph.D. Thesis, Northeastern University.

[34] Á. Seress and I. Weisz, *PERM: A Program Computing Strong Generating Sets*, this volume.

[35] C.C. Sims, *Computation with permutation groups*, Proc. Second Symposium on Symbolic and Algebraic Manipulation (S.R. Petrick, ed.), ACM Press, 1971, pp. 23–28.

College of Computer Science, Northeastern University, Boston, MA 02115

E-mail address: gene@ccs.neu.edu and laf@ccs.neu.edu