Fast Monte Carlo Algorithms for Permutation Groups

László Babai[1,2], Gene Cooperman[3], Larry Finkelstein[3], Eugene Luks[4], and Ákos Seress[5]

[1]Department of Computer Science
University of Chicago
Chicago, Illinois 60637

[2]Department of Algebra
Eötvös University
Budapest, Hungary H-1088

[3]College of Computer Science
Northeastern University
Boston, Mass. 02115

[4]Department of Computer Science
University of Oregon
Eugene, Oregon 94705

[5]Department of Mathematics
The Ohio State University
Columbus, Ohio 43210

Fast Monte Carlo Algorithms for Permutation Groups

László Babai[1,2], Gene Cooperman[3], Larry Finkelstein[3], Eugene Luks[4], and Ákos Seress[5]

[1]Department of Computer Science
University of Chicago
Chicago, Illinois 60637

[2]Department of Algebra
Eötvös University
Budapest, Hungary H-1088

[3]College of Computer Science
Northeastern University
Boston, Mass. 02115

[4]Department of Computer Science
University of Oregon
Eugene, Oregon 94705

[5]Department of Mathematics
The Ohio State University
Columbus, Ohio 43210

Abstract

We introduce new, elementary Monte Carlo methods to speed up and greatly simplify the manipulation of permutation groups (given by a list of generators). The methods are of a combinatorial character and use elementary group theory only. The key idea is that under certain conditions, "random subproducts" of the generators successfully emulate truly random elements of a group.

We achieve a nearly optimal $O(n^3 \log^c n)$ asymptotic running time for membership testing, where n is the size of the permutation domain. This is an improvement of two orders of magnitude compared to known elementary algorithms and one order of magnitude compared to algorithms which depend on heavy use of group theory.

An even greater speedup is achieved for *normal closures*, a key ingredient of algorithms in computational group theory, now constructible in Monte Carlo time $O(n^2 \log^c n)$, i.e. essentially linear time (as a function of the input length).

As a by-product, for low rank systems of linear equations we achieve a speedup over Gaussian elimination.

On the other hand, some of the new techniques are so general, that they allow polynomial time implementations in the very general model of "black box groups" (group operations are performed by an oracle). In particular, the normal closure algorithm leads to polynomial time Monte Carlo algorithms to test solvability and nilpotence of such groups, and a number of further applications to matrix groups.

It should be stressed that our randomized algorithms are not heuristic: the probability of error is guaranteed not to exceed a bound $\varepsilon > 0$, prescribed by the user. The cost of this requirement is a factor of $|\log \varepsilon|$ in the running time.

1. Introduction

1.1. A brief summary

We introduce new, simple, and efficient Monte Carlo algorithms to perform basic manipulation of permutation groups.

[†]Research partially supported by NSF Grant CCR-8903952.

The permutation groups considered are given by a list of generators. The basic task is to transform this into a data structure called a *strong generating set* (SGS). Given an SGS, one can rapidly determine the order of G and decide membership in G.

Strong generating sets were introduced by Sims [Si] in the 60's and have been used extensively in computational group theory. Variants of Sims's elementary algorithm for construction of a SGS have been known to have $\Theta(n^5)$ worst case behavior, where n is the cardinality of the permutation domain. It was only with extensive references to consequences of the classification of finite simple groups that a much more involved algorithm of Babai, Luks, and Seress reduced the worst case running time to $O^{\sim}(n^4)$ [BLS2] where the tilde refers to a $(\log n)^c$ factor.

Using randomization in a novel way, we achieve considerable further speedup in basic permutation group manipulation. We are able to construct an SGS in Monte Carlo time $O^{\sim}(n^3)$, nearly matching the running time of Gaussian elimination which is a (very) particular case. As a consequence, we can determine the order of G in Monte Carlo time $O^{\sim}(n^3)$ and subsequently perform membership tests in $O(n^2)$ time per test.

The most impressive speedup is achieved for *normal closures,* a key ingredient of algorithms in computational group theory, previously thought to depend on prior construction of an SGS. We can now construct normal closures in Monte Carlo time $O^{\sim}(n^2)$, i.e. essentially *linear time.* (The input typically consists of $O(n)$ permutations and thus has length $O(n^2)$.)

As a by-product, for low rank systems of linear equations we achieve a speedup over Gaussian elimination.

On the other hand, some of the new techniques are so general, that they allow polynomial time implementations in "black box groups" (group elements are encoded by strings of uniform length, and the group operations are performed by an oracle). In particular, the normal closure algorithm leads to polynomial time Monte Carlo algorithms to test solvability and nilpotence of black box groups (given, as all our groups, by a list of generators). These results apply to matrix groups over finite fields (since their elements are encoded automatically as strings of uniform length; the oracle is replaced by matrix multiplication/inversion routines). Nontrivially, the results also apply to finite matrix groups over the rationals and over algebraic number fields [BBR]. (Finiteness of such groups can be tested in polynomial time [BBR].) The normal closure algorithm is used extensively in [BeB], resulting in polynomial time *Las Vegas* algorithms to find the order and structural elements such as a composition chain in finite matrix groups over an algebraic number fields, and to uncover a considerable portion of the structure in matrix groups over finite fields. We should mention that E. M. Luks [Lu] found polynomial time *deterministic* algorithms to test solvability and nilpotence of matrix groups. He also found the order and many structural elements of solvable matrix groups assuming (as does [BeB]) in the case of finite fields the tractability of certain arithmetic problems (factoring integers of the form $p^k - 1$ and discrete logarithm), inherent already in the abelian case.

On the other hand, no deterministic polynomial time solution of the normal closure problem has yet been found for matrix groups.

Our algorithms are nearly optimal, as indicated, for large classes of permutation groups. For the important subcase of groups with *small* (polylogarithmic size) *bases*, nearly linear time Monte Carlo algorithms have been found, using entirely different methods. These are described in a separate paper [BCFS].

1.2. Strong generators, normal closure

Given a chain $G = G_0 \geq G_1 \geq \ldots \geq G_m = 1$ of subgroups of a finite group G, a *strong generating set* (SGS) with respect to this chain is a set $T \subset G$ such that $T \cap G_i$ generates G_i for each i. Sims [Si] introduced this concept with respect to the point stabilizer subgroup chain for finite permutation groups acting on n points. In that case G acts on the set $\Omega = \{1, \ldots, n\}$, and G_i is the pointwise stabilizer of the initial segment $\{1, \ldots, i\}$ (and $m \leq n$).

Strong generatings sets provide the basic data structure in essentially all polynomial time algorithms for permutation groups.

Let G be a group given by a list of s generators. The first asymptotically analyzed variant of Sims's SGS algorithm [FHL] ran in time $O(n^6 + sn^2)$. This was subsequently improved by Knuth [Kn] and Jerrum [Je] to $O(n^5 + sn^2)$. Knuth's version follows Sims's most closely. It is also shown in [Kn] that if G is given by a certain randomized list, then the algorithm actually requires $\Omega(n^5)$ time on average.

Subsequent asymptotic improvements are most conveniently stated in the $O^\sim$ ("soft-oh") notation which we define.

Notation. Let f, g be two real-valued functions over the positive integers. We say that $f(n) = O^\sim(g(n))$ if there exists a positive constant c such that for every sufficiently large n,

$$f(n) \leq g(n)(\log n)^c.$$

If f and g are functions of more than one variable, the argument of the log will be the product of those variables, e. g.

$$f(n, s) \leq g(n, s)(\log(ns))^c.$$

The notorious n^5 bound was brought down by Babai, Luks and Seress [BLS2] in an $O^\sim(n^4+sn^2)$ algorithm which digs deeply into the normal structure of G and requires several consequences of the classification of finite simple groups for its analysis. This algorithm was recently improved to $O^\sim(sn^3)$ ([BLS3], [BLS4]). We note that it is reasonable to assume $s \leq n$ and in fact often s is much smaller ($s = 2$ is a frequent case).

In the present paper we achieve considerable *Monte Carlo speedups* over the previously known algorithms. (Monte Carlo algorithms will be defined in the next section.) A further advantage of the new algorithms is that their analysis is *completely elementary*; and purely combinatorial in nature.

One of the first difficulties to overcome is the growth of the number of generators for subgroups constructed. Although any set of $\geq 2n$ generators of a permutation group of degree n is redundant [Ba2], we may not know which generators to delete. Our first result solves this problem:

Theorem 1.1. *One can construct $O(n)$ generators for a permutation group of degree n from an initially given set of s generators in Monte Carlo time $O(sn \log n)$.*

Henceforth when using the expression "we construct a subgroup", it is understood that we construct a list of $O(n)$ generators for that subgroup; and, if not specified otherwise, the input groups are also assumed to be given by lists of $O(n)$ generators.

Here are our *main results* for permutation groups $G \leq \mathrm{Sym}(\Omega)$ where $|\Omega| = n$.

Theorem 1.2.

(i) Given a group G and a subgroup H, each by a list of s generators, we construct the normal closure of H in G in Monte Carlo time $O^\sim(n^2 + ns)$;

(ii) Given a group G by a list of s generators and a point $x \in \Omega$, we construct the stabilizer of x in Monte Carlo time $O(ksn \log n)$, where k is the length of the orbit of x;

(iii) Given a group G by a list of s generators, we construct the point stabilizer chain (SGS) in Monte Carlo time $O^\sim(n^3 + ns)$. We can determine the order of G within the same time bound, and set up Jerrum's "labeled branching" data structure which supports membership tests for G in $O(n^2)$ time per test. — The algorithm can be upgraded to Las Vegas, running in time $O(n^4 + sn \log n)$ (no hidden log factors).

(Las Vegas algorithms are randomized algorithms that never err. See Section 1.3 for more about this concept.)

Using Jerrum's "labeled branching" data structure [Je] we keep the *space* required by these algorithms at $O^{\sim}(n^2 + ns)$, i.e. linear compared to a typical input of $O(n)$ permutations.

The $O^{\sim}(n^3)$ time bound is optimal within a polylog factor in the sense that it includes Gaussian elimination as a very special case.

The Las Vegas upgrade is obtained by combining our Monte Carlo algorithm with the $O(n^4)$ deterministic strong generation test of Cooperman and Finkelstein [CF1]. This might be significant in implementations because it allows to take advantage of favorable circumstances by early termination.

Previous polynomial-time randomized algorithms for permutation groups have been largely limited to heuristic algorithms which were difficult to analyze. Faster base change algorithms [Ba1, CFS] were some of the first algorithms to provide a rigorous analysis for the use of uniformly distributed random group elements. However, no method is known to generate uniform random elements of G without first constructing a strong generating set. (A random walk method exists to construct nearly uniformly distributed random elements in a group directly, in polynomial time [Ba3]. While near-uniformity would suffice for our purposes, the time bounds in [Ba3] are not competitive compared to those presented here.)

The main idea elaborated in the present paper is that under certain circumstances, *products of random subsets of the generators can successfully emulate truly random elements* (see Section 2.2). This idea seems to have appeared first in the superfast Las Vegas handling of the symmetric group by Babai, Luks and Seress [BLS2]. There, it is shown that $O(\log n)$ random subproducts are likely to generate a subgroup H with orbit structure identical to G. This yields in particular $O(\log n)$ generators of a doubly transitive subgroup in Monte Carlo time $O(|S|n \log n)$ assuming $G = \langle S \rangle$ is doubly transitive. Both the number of generators thus obtained and the time required are almost an order of magnitude smaller than the $O(n)$ generators obtained deterministically in time $O(|S|n^2)$.

1.3. Reliability, and spreader sets of permutations

A *Monte Carlo algorithm* is a randomized algorithm that may give a wrong answer or report failure with *guaranteed* small probability.

We note that many heuristic randomized algorithms are known to perform well in the practice of computational group theory. We should stress here that our bounds on the probability of error/failure are *guaranteed* and are *prescribed by the user*.

A Las Vegas algorithm is a Monte Carlo algorithm that never gives a wrong answer. (The term was introduced in [Ba1], cf. [Joh],p. 437.)

The *running time* of a Monte Carlo algorithm is defined with respect to error/failure probability 1/4. The penalty for requiring the probability to be $\leq \varepsilon$ is an increase by a factor of $O(\log(1/\varepsilon))$ in the timing. (Run the algorithm many times and take majority vote.)

Our Monte Carlo algorithms are not Las Vegas, so we shall never be sure of the correctness of the output. The probability of error, however, can be made less than $\exp(-(\log n)^C)$ for any constant C within the time bounds stated.

Combining our algorithm with the deterministic $O(n^4)$ time "strong generating test" of Cooperman and Finkelstein [CF1], we obtain an $O(n^4)$ Las Vegas SGS algorithm, which requires elementary group theory only. (Note, however, that now, an $O^{\sim}(sn^3)$ deterministic SGS algorithm also exists; but, that algorithm invokes the classification of finite simple groups [BLS4].) [BeB] provides further examples where the Monte Carlo normal closure routine is employed in a Las Vegas algorithm (determining the order and a composition chain in finite matrix groups over algebraic number fileds).

It would be desirable to *increase the reliability* of our algorithms to exponential without the loss of an extra order of magnitude in running time. This depends on whether or not an explicit construction can be found to replace a set of random permutations used in the algorithm. We state this problem below.

Let $M = \{1, \dots, m\}$, $\overline{M} = M \cup \{0, m+1\}$, $H \subseteq M$, and $\overline{H} = H \cup \{0, m+1\}$. For $x \in H$, let

$$\delta_H(x) = \min_{y \in \overline{H} \setminus \{x\}} |x - y|.$$

(Distance to nearest neighbor.) Define $\delta(H) = \sum_{x \in H} \delta_H(x)$ and $\text{spread}(H) = \delta(H)/m$. Obviously, $0 \leq \text{spread}(H) \leq 1$. We use this quantity, the *spreading factor of* H, to measure how evenly the set H is spread out within M. We say that H is *ε-spread* if $\text{spread}(H) \geq \varepsilon$.

Definition 1.3. *Let $P = \{\pi_1, \dots, \pi_r\}$ be a set of permutations of the set $M = \{1, 2, \dots, m\}$. We call P an (ε, m)-spreader if for every nonempty set $H \subseteq M$, at least one of the sets H^{π_j} is ε-spread $(j = 1, \dots, r)$. The greatest such ε is the* spreading factor *of P.*

Theorem 1.4. *With large probability, a set of $r = O(\log m)$ random permutations is a (ε, m)-spreader, where ε is an absolute constant.*

We shall show in Section 3 that $\varepsilon \approx 1/15$ can be achieved, but this result is probably not the strongest possible. We conjecture that the spreading factor of $O(\log m)$ random permutations is considerably better, possibly even $\varepsilon \approx 1/4$.

Spreaders will be a key component of some of our algorithms, including the superefficient normal closure. The reliability of the random spreaders given by Theorem 1.4 is $1 - m^{-c'}$ for an arbitrary constant c'. Because of the crucial role they play, it would be of great importance to construct *guaranteed spreaders* either by way of a Las Vegas construction or by an explicit construction, or at least to improve their reliability to exponential.

1.4. Black box groups

Some of the new algorithms are so general, they work even in the "black box group" model. In this model, introduced in [BSz], group elements are encoded by strings of uniform length n, and group operations are performed by an oracle (the black box). If the group G is given by the strings representing a set of generators, then G is called a *black box group*. Note that the order of such a group is $|G| \leq 2^n$.

The most important implementations of the black box group model are *matrix groups* over finite fields and over algebraic number fields (see the comments in Section 1.1). Over finite fields, even the membership problem is unlikely to be solvable in polynomial time since, even in the case of 1×1 matrices, membership is closely related to finding discrete logarithms, a task not believed to be solvable in polynomial time. (However, using an oracle for discrete log, a lot can be achieved in polynomial time; cf. the comments in Sec. 1.1.)

While the rudimentary task of membership testing remains intractable, some global structural properties of the group can be determined in Monte Carlo polynomial time, even in the general context of black box groups.

Theorem 1.5. *Let G be a black box group with elements encoded as strings of length n. Suppose G is given by a list of s generators. Then the number of generators can be reduced to $O(n)$ by a Monte Carlo algorithm consisting of $O(s \log n)$ group operations. Moreover, if both G and a subgroup H are given by $O(n)$ generators, then the normal closure of H in G can be constructed by a Monte Carlo algorithm consisting of $O^{\sim}(n)$ group operations.*

Corollary 1.6. *Let G be as in Theorem 1.5. Then one can perform each of the following tasks in Monte Carlo polynomial time: (a) construct the commutator subgroup; (b) decide whether or not G is (b1) solvable; (b2) nilpotent. If G is solvable, the commutator chain can be constructed; and if G is nilpotent, the descending central series can be constructed.*

The proof of these results is a simple combination of the algorithm to reduce the number of generators and the fast normal closure algorithm (cf. Sections 2.2 and 3). It uses the observation that the length of a chain of subgroups of a black box group G is $\leq n$ (since $|G| \leq 2^n$).

1.5. Pruning redundant systems of linear equations

There is a curious consequence of the generator pruning algorithm to low rank matrices. If a matrix over a field F has m rows and rank r then Gaussian elimination requires $\Theta(mr)$ row operations to determine the rank or to solve a system of linear equations.

If $r = o(m)$ but r itself is not bounded, we can achieve considerable savings by first replacing the m rows by a set of $O(r)$ rows using reduction of generators (Theorem 2.7). With large probability, the rows obtained will generate the same row space. The new rows are obtained as sums of subsets of the original rows and will require $O(m \log r)$ row operations to compute. Subsequently we can complete the work by Gaussian elimination, using $O(r^2)$ row operations.

Corollary 1.7. *A system of m linear equations of rank r can be solved (or found inconsistent) by a Monte Carlo algorithm using $O(r^2 + m \log r)$ row operations.*

(Note that Gaussian elimination would require $O(mr)$ row operations.)

A "repeated doubling" trick achieves this complexity even if no a priori estimate of r is available.

2. Random subproducts

In this section, random subproducts are defined and our first technical goal, reduction of the number of generators, is achieved. This algorithm works in general even for black box groups. The application to low rank systems of equations stated in Section 1.5 follows immediately. As another application, we achieve a second main goal, efficient computation of generators for a stabilizer subgroup in a permutation group.

2.1. Estimating sums of dependent random variables

In the algorithms here and throughout the paper, progress will be measured by the sum of random *dependent* (0,1)-variables with guaranteed lower bounds on their expectations, given any history of preceding steps. In estimating the probability that sufficient progress has been made, the following lemma will come handy. The result appears to be folklore but we have not found a convenient reference.

Let $\xi_1, \xi_2, \ldots$ be a sequence of random variables. A *history* for ξ_i is a condition of the form $U_i = \{\xi_1 = \alpha_1, \ldots, \xi_{i-1} = \alpha_{i-1}\}$. We shall say that ξ_i has expectation $\geq p$ *under any history* if $E(\xi_i | U_i) \geq p$ for any history U_i. We say that the sequence $\{\xi_i\}$ dominates the sequence $\{\zeta_i\}$ if $\xi_i \geq \zeta_i$ (with probability 1). A *Bernoulli trial* with expectation p is a $(0,1)$-variable τ such that $\text{Prob}(\tau = 1) = p$.

Lemma 2.1. *Let $0 \leq p \leq 1$. Let $\xi_1, \xi_2, \ldots$ be a sequence of random $(0,1)$-variables such that each ξ_i has expectation $\geq p$ under any history. Then there exists a sequence $\tau_1, \tau_2, \ldots$ of independent Bernoulli trials with expectation p, dominated by $\xi_1, \xi_2, \ldots$.*

Proof. We use the sequence $\xi_1, ..., \xi_i$ to construct the variable τ_i. Given a history U_i of $\xi_1, \ldots, \xi_{i-1}$ as above, let $\text{Prob}(\xi_i = 1 | U_i) = q$. By assumption, $q \geq p$. Let η_{U_i} be a Bernoulli trial with

expectation $r = p/q$. We assume that the η_{U_i} are independent of each other and of all the ξ collectively. Now we set $\tau_i := \xi_i \eta_{U_i}$.

It is clear that τ_i is a random $(0,1)$-variable and $\tau_i \leq \xi_i$ (always). Moreover, $E(\tau_i|U_i) = E(\xi_i|U_i)E(\eta_{U_i}) = q \cdot (p/q) = p$. It is immediate then that $E(\tau_i|V_i) = p$ holds under any combined history V_i of the ξ_j and the η_{U_j} $(j \leq i-1)$, and therefore the same holds under any history of the τ_j. Consequently the τ_i are fully independent (their distribution does not depend on their histories). We also have $E(\tau_i) = p$. □

It follows that Chernoff's powerful bound for the sum of independent Bernoulli trials can be applied directly to our dependent variables.

Corollary 2.2. *Under the conditions of Lemma 2.1, the following inequality holds for any positive integer t and any $\varepsilon > 0$.*

$$\text{Prob}\left(\sum_{i=1}^{t} \xi_i \leq (1-\varepsilon)pt\right) \leq e^{-\varepsilon^2 pt/2}.$$

Proof. This is what Chernoff's bound asserts for the τ_i in the place of the ξ_i. □

2.2. Reduction of generators

Definition 2.3. *A subproduct of a sequence of group elements $(g_1, g_2, \ldots, g_k)$ is a product of the form $g_1^{e_1} g_2^{e_2} \cdots g_k^{e_k}$ where $e_i \in \{0,1\}$. To obtain a random subproduct, select the e_i independently from the uniform distribution over $\{0,1\}$. A random subproduct of length ℓ is obtained as follows: first select a subsequence of length ℓ of the g_i uniformly from the $\binom{k}{\ell}$ possibilities; then form a random subproduct of them (with respect to the inherited ordering). (Note that in this terminology, a random subproduct of length ℓ is expected to be a product of $\ell/2$ terms.)*

Suppose now that G acts on the set Ω and assume that $K \subset \Omega$ is *not* stabilized by G. Then, as observed in [BLS2, section 6.2], with probability $\geq 1/2$, K is not stabilized by a random subproduct of the generators of G. The following statement is easily seen to be equivalent to this observation.

Lemma 2.4. *If K is a proper subgroup of the finite group G and $(g_1, g_2, \ldots, g_k)$ are generators of G, then for a random subproduct $g = g_1^{e_1} g_2^{e_2} \cdots g_k^{e_k}$, we have* $\text{Prob}(g \notin K) \geq 1/2$.

Proof. For completeness, we adapt the proof from [BLS2, section 6.2]. Let j be the largest subscript such that $g_j \notin K$. Then $g = a g_j^{e_j} b$ where a depends on $e_1, \ldots, e_{j-1}$ only, and $b \in K$. Let us now fix the values of all the e_i except e_j. There are two cases to consider. If $a \in K$ then $e_j = 1$ implies $g \notin K$; if $a \notin K$ then $e_j = 0$ implies $g \notin K$. Thus in either case, $\text{Prob}(g \notin K) \geq 1/2$. □

Theorem 2.5. *Let $G = \langle S \rangle$ be a finite group. Let $|S| = s$, and let L be a known upper bound on the length of all subgroup chains in G. Then, using $O(s \log L)$ group operations, one can with fixed but high probability find a generating set S' with $|S'| = O(L)$.*

Proof. Assume $s > 4L$ or else there is nothing to do. Starting from $S' = \emptyset$, successively place in S', cL random subproducts each of length s/L. We claim that after this phase, with high probability, all but L of the elements of S are in $\langle S' \rangle$.

Here is a heuristic reason. If the claim were false then, for a randomly chosen $g \in S$, we would have $\text{Prob}(g \notin \langle S' \rangle) > L/s$. So each of the cL random subproducts would have probability at least $\alpha := (1 - (1 - L/s)^{s/L})/2 \approx (1 - 1/e)/2 \approx 0.316$ of not being in the previously constructed $\langle S' \rangle$. For sufficiently large c, this would violate the bound L on the length of subgroup chains.

The reason why this argument is not precise is that under a condition on the outcome (such as the condition that that we end up having many elements left out of $\langle S' \rangle$), our choices are no longer uniform and independent.

An application of Corollary 2.2 will help make this argument precise. Let $g_1, g_2, \ldots$ denote the random subproducts constructed by the procedure, and let G_i denote the subgroup generated by $\{g_1, \ldots, g_i\}$. Let ξ_i be the indicator variable taking value $\xi_i = 1$ if either $|S \setminus G_i| \leq L$ or G_{i-1} is a proper subgroup of G_i; else set $\xi_i = 0$. Now it is clear that for every i, $\text{Prob}(\xi_i = 0 \mid U_i) \leq \alpha$, where U_i is a condition fixing the values of $g_1, \ldots, g_{i-1}$ arbitrarily.

It follows that the ξ_i satisfy the conditions of Lemma 2.1 and therefore by Corollary 2.2 we obtain that for any constant $c > 1/(1-\alpha) \approx 1.462$, it is exponentially likely (as a function of L) that $\sum_{i=1}^{cL} \xi_i > L$. But then, this inequality implies that $|S \setminus G_{cL}| \leq L$, since a subgroup chain cannot grow more than L times..

Now let us continue the description of the procedure. Our initial phase capitalized on the intuition that initially, even relatively short random subproducts have a fair chance of increasing the current subgroup $\langle S' \rangle$.

After the initial round discussed above, we continue augmenting S' with subproducts of S of increasing lengths for $\log L$ more rounds. (We can suppose that $L = 2^k$ for some integer k). Let T_i denote S' after the ith round, so we are very likely to start with T_0 satisfying $|S \setminus \langle T_0 \rangle| \leq L$. In the ith round, the *goal* is to reduce the number $d_i = |S \setminus \langle T_i \rangle|$ below $L/2^i$. To this end, we fix a new constant c' and in the ith round add $c'L/2^i$ random subproducts of S of length $2^{i-1}s/L$.

Suppose our goal has been met in round $i-1$: $d_{i-1} < L/2^{i-1}$. We claim that under this condition, $\text{Prob}(d_i < L/2^i) > 1 - e^{-c'c''L/2^i}$ for some constant $c'' > 0$ not depending on c'.

Let the random subproducts generated in the ith round be $g_1, g_2, \ldots$. Set $H_j = \langle T_{i-1}, g_1, \ldots, g_{j-1} \rangle$ (the current group) and $R_j = S \setminus H_j$ (the generators remaining outside).

Let ξ_j denote the indicator of the event that either $|R_j| < L/2^i$ or g_j (considered as a product of elements of S) contains *exactly one* element of R_j. In the former case the goal of the ith phase has already been achieved; in the latter, $|R_{j+1}| = |R_j| - 1$. Clearly, if $\sum_{j \leq c' l_G/2^i} \xi_j > l_G/2^i$ then fewer than $l_G/2^i$ elements of S are not in T_i. Hence Corollary 2.2 implies that in order to prove our claim, it is enough to give a positive constant lower estimate for $\text{Prob}(\xi_j = 1)$.

Let B_j denote the set of those elements of S selected in order to create the random subproduct g_j ($|B_j| = 2^{i-1}s/L$). If $L/2^{i-1} > |R_j| \geq L/2^i$ then the probability that B_j contains exactly one element of R_j is

$$|R_j| \binom{s - |R_j|}{2^{i-1}s/L - 1} \Big/ \binom{s}{2^{i-1}s/L} = |R_j| \frac{2^{i-1}}{L} \prod_{k=0}^{2^{i-1}s/L-1} \frac{s - |R_j| - k}{s-k} > \frac{1}{2}\left(1 - \frac{L/2^{i-1}}{s - 2^{i-1}s/L}\right)^{2^{i-1}s/L} > \frac{1}{2e^2}.$$

If B_j contains a unique element of $h \in R_j$ then with probability $1/2$, h will occur in g_j. Hence $\text{Prob}(\xi_j = 1) > 1/(4e^2)$.

The bounds we obtained for error probabilities for each round form a geometric progression dominated by the last term which is an arbitrarily small constant, depending (exponentially) on our choice of c'. □

An alternative algorithm uses spreaders (cf. Section 1.3) with similar efficiency (cf. Section 3.2.) In applying this result to permutation groups, we require the following lemma from [Ba2], [CST].

Lemma 2.6. *Any subgroup chain in S_n has length at most $2n-3$.*

Consequently, $L = 2n$ is a valid bound for permutation groups of degree n.

When applying Theorem 2.5 to black box groups where elements are encoded by binary strings of length n, we may use the obvious upper bound $\log |G| \leq n$ on the length of subgroup chains.

2.3. Group closure systems

The proof of Theorem 2.5 actually yields the following more general result. Let U be an arbitrary set and Φ a family of subsets of U closed under intersections, such that $U \in \Phi$. We call the pair (U, Φ) a *closure system*. The members of Φ are called *closed sets*; and the closure $\mathrm{cl}(S)$ of a subset S is the intersection of all closed sets containing S. The set S is said to generate its closure.

We call (U, Φ) a *group closure system*, if in addition, U is a group under a certain operation, and all closed subsets are subgroups. (Note that not all subgroups need to be closed.)

Examples of group closure systems abound: consider groups with any sublattice of their subgroups, e.g. the normal subgroups; or rings with any sublattice of their subrings, e.g. their ideals; or vector spaces and their subspace lattice.

Theorem 2.7. *Let (U, Φ) be a group closure system. Assume that U is generated by a finite set S, $|S| = s$, and let L be a known finite upper bound on the length of all chains of closed sets. Then, using $O(s \log L)$ group operations, one can with fixed but high probability find a generating set S' with $|S'| = O(L)$.*

The proof is identical with that of Theorem 2.5. An application of this result to the subspaces of the row space of a matrix justifies the argument given in Section 1.5. (Take L to be the rank of the matrix.)

2.4. Schreier generators and point stabilizers

Our next task, of great importance for permutation groups, is to construct generators for subgroups. The common technique is Schreier generators, which we now define.

Definition 2.8. *A (right) transversal of the subgroup H in the group G is a complete set of (right) coset representatives of H in G.*

Definition 2.9. *Let $G = \langle S \rangle$ be a group, H a subgroup, and T a transversal of H in G. For each $t \in T$ and $g \in S$, let t' be the unique element of T such that $tg \in Ht'$. Then the element tgt'^{-1} is called a Schreier generator of H.*

The total number of Schreier generators is $|S| \cdot |T|$.

Fact 2.10. *The Schreier generators generate H.*

See [Ha, Lemma 7.2.2] for a proof.

The difficulty with (repeatedly) using Schreier generators is in their rapidly growing number. This obstacle is removed using Theorem 2.5: first we construct the Schreier generators; then we reduce their number to $O(L)$. All this takes $O(|S||T| \log L)$ group operations, assuming we can quickly determine t' from tg ($t \in T, g \in S$).

We apply this result to the point stabilizers in permutation groups.

Theorem 2.11. *Let $G = \langle S \rangle \leq \mathrm{Sym}(\Omega)$ be a permutation group of degree $n = |\Omega|$, $|S| = s$, and let $x \in \Omega$. Then we can find a set of $O(n)$ generators of the stabilizer subgroup G_x in Monte Carlo time $O(nsk \log n)$ where k is the length of the G-orbit of x. The storage requirement is $O(n^2 + sn)$.*

Proof. To construct a transversal of G_x in G, we consider the action of the generators of G on the orbit x^G. This is described by a graph which has sk (directed) edges, labeled by generators. A coset representative corresponding to point y of this orbit can be obtained as the product of labels along the $x - y$ path in a spanning tree. This requires a total of $O(sk + kn)$ time.

Now the sk Schreier generators are constructed in time $O(nsk)$. Finally the reduction of their number to $O(n)$ is accomplished via Theorem 2.5 in time $O(nsk \log n)$.

Computing all Schreier generators at the same time requires $O(skn)$ storage which, at a typical application with $s = \Theta(n), k = \Theta(n)$, gives the prohibitively large $\Theta(n^3)$. In order to avoid this blowup, we store only the elements of the transversal for G_x in G. Whenever a Schreier generator is needed (as an element of a random subproduct in the reduction procedure of Theorem 2.5), we compute it from the transversal and then we discard it. This modification multiplies the running time only by a constant factor and reduces the storage requirement to $O(n^2)$ (besides the $O(sn)$ storage required for S). □

It follows that we can construct the *entire stabilizer chain* in Monte Carlo time $O(n^4 \log n)$, by repeated application of Theorem 2.11, using the trivial bound $k \leq n$.

We have thus obtained a strikingly simple SGS construction in $O^{\sim}(n^4)$ Monte Carlo time. (An analogous deterministic bound but with a few extra log factors, has previously been reached using the classification of finite simple groups [BLS2].) Combined with the $O(n^4)$ deterministic strong generation test of Cooperman and Finkelstein [CF1], this Monte Carlo algorithm is upgraded to Las Vegas without loss of efficiency.

We mention that with a slightly different application of random subproducts, one can achieve a little better, $O(n^4)$ Las Vegas time [CF2].

3. Asymptotically very fast Monte Carlo normal closure

The normal closure of a subgroup H in a group G is the smallest normal subgroup of G containing H.

The methods of the previous section can be used to construct the normal closure in time $O(n^3)$ in permutation groups of degree n.

In this section we shall use spreaders (cf. Section 1.3) to achieve an even faster algorithm.

Theorem 3.1. *If $H \leq G \leq S_n$ then the normal closure of H in G can be constructed in Monte Carlo time $O(n^2 \log^4 n)$.*

Here we assume that both G and H are given by lists of $O(n)$ generators. This assumption is justified in view of Theorem 2.5 (cf. Lemma 2.6).

The key concept responsible for the speedup is *random prefixes.* The idea is that once we computed a long product of generators, it seems wasteful to use it only once. Since all the prefixes have been computed along the way, we wish to use the prefixes as well. We randomize the prefixes in two different ways. First, we randomize the order in which the elements are multiplied together; and then we select at random which prefix we want to use.

In order to ensure that a random prefix will indeed behave like a random subproduct in the sense that it will have a good chance of adding something to our current subgroup, we require that the order in which the elements are arranged spreads out the generators which are not in our current subgroup. Since we don't know which are these generators, instead we require that every nonempty set be spread out, i.e. we need a spreader set of permutations (cf. Section 1.3).

3.1. Random spreaders

We use the terminology and notation introduced in Section 1.3.

Lemma 3.2. *Let H be a random subset of size $h \neq 0$ of $M = \{1, \ldots, m\}$ and $1/30 \leq \varepsilon < 1/15$. Then* $\mathrm{Prob}(\mathrm{spread}(H) \geq \varepsilon) > 1 - c^h$, *where* $c = (15\varepsilon)^{1/4}$.

Proof. If $|H| \geq \varepsilon(m+1)$ then $\mathrm{spread}(H) \geq |H|/(m+1) \geq \varepsilon$ with probability 1.

If $1 \leq h \leq 4$ and $h < \varepsilon(m+1)$ then we compute the ratio q of h-element subsets $a_1 < a_2 < \cdots < a_h$ of M such that $a_i - a_{i+1} \geq \lceil \varepsilon(m+1)/h \rceil$ for $1 \leq i \leq h+1$, with $a_0 := 0$ and $a_{h+1} := m+1$.

For such sets H, $\mathrm{spread}(H) \geq h\lceil \varepsilon(m+1)/h \rceil \geq \varepsilon$. Introducing the notation $d = \lceil \varepsilon(m+1)/h \rceil - 1$,

$$\mathrm{Prob}(\mathrm{spread}(H) \geq \varepsilon) \geq q = \binom{m-(h+1)d}{h} \Big/ \binom{m}{h} > \left(\frac{m-h+1-(h+1)d}{m-h+1}\right)^h$$
$$= \left(1 - \frac{(h+1)d}{m-h+1}\right)^h > 1 - h\frac{(h+1)d}{m-h+1} > 1 - \frac{(h+1)\varepsilon(m+1)}{m-h+1} >$$
$$1 - \frac{(h+1)\varepsilon(m+1)}{(1-\varepsilon)(m+1)} > 1 - (h+1)\varepsilon(15/14) > 1 - (15\varepsilon)^{h/4}.$$

In the case $5 \leq h < \varepsilon(m+1)$, we consider H as the range of an injective function f from $H_0 := \{1, 2, ..., h\}$ to M. We also define $\overline{H_0} := H_0 \cup \{0, h+1\}$ and extend f by setting $f(0) := 0$ and $f(h+1) := m+1$.

Let $k := \lfloor h/2 \rfloor$. We estimate the probability of the event $A(c)$ that there exist k distinct elements of $\overline{H_0}$, $a_{1,1}, ..., a_{1,k_1}, a_{2,1}, ..., a_{2,k_2}, ..., a_{l,1}, ..., a_{l,k_l}$, $2 \leq k_i \leq 3$ for $1 \leq i \leq l$, $\sum k_i = k$, such that $f(a_{i,j}) < f(a_{i,j+1}) < f(a_{i,j}) + c(m+1)$ for all $1 \leq i \leq l$, $1 \leq j \leq k_i - 1$. Such k elements can be chosen

$$\binom{h+1}{l}\binom{l}{k-2l}\frac{(h+2-l)!}{(h+2-k)!}$$

ways, since the elements $a_{1,1}, a_{2,1}, ..., a_{l,1}$ are from $\overline{H_0} \setminus \{h+1\}$; from this set, we have to choose a $k-2l$-element subset to determine those $a_{i,1}$ which have $k_i = 3$; finally, fixing the two subsets of the $a_{i,1}$, there are $(h+2-l)(h+1-l)\cdots(h+3-k)$ choices for the remaining $a_{i,j}$.

Fixing the set of $a_{i,j}$, we can define f by first choosing the (random) values $f(a_{i,1})$ for $1 \leq i \leq l$ and then continuing with the choice of $f(a_{i,j})$ for $j \geq 2$. The probability that $f(a_{i,j+1})$ falls between $f(a_{i,j})$ and $f(a_{i,j}) + c(m+1)$ is at most $c(m+1)/(m+1-(k-1))$, hence

$$\mathrm{Prob}(A(c)) < \binom{h+1}{l}\binom{l}{k-2l}\frac{(h+2-l)!}{(h+2-k)!}\left(\frac{c(m+1)}{m+1-k}\right)^{k-l}.$$

Since $k \leq h/2 \leq \varepsilon(m+1)/2 \leq (m+1)/30$, $(m+1)/(m+1-k) \geq 30/29$. We claim that

$$\binom{h+1}{l}\binom{l}{k-2l}\frac{(h+2-l)!}{(h+2-k)!} < \left(\frac{29h}{4}\right)^{k-l}, \qquad (*)$$

and so $\mathrm{Prob}(A(c)) < (15ch/2)^{k-l}$.

For $h \leq 200$, $(*)$ can be checked by computer. For larger values, we use Stirling's formula. Note that the sequence $\sqrt{2\pi n}(n/e)^n/n!$, $n = 0, 1, ...$ is monotone increasing with limit 1, so substituting the appropriate $\sqrt{2\pi n}(n/e)^n$ for the factorials gives an upper estimate for the left-hand-side of $(*)$.

Using that for $1/3 \leq x \leq 1/2$

$$(3x-1)^{3x-1}(1-2x)^{1-2x} > 4^x(29e/8)^{x-1},$$

$(3l-k)^{3l-k}(k-2l)^{k-2l}$ can be estimated from below by $k^l 4^l (29e/8)^{l-k}$. Also, $(h+2-k)^{h+2-k} > e^2(h/2)^{h+2-k}$. Using these estimates, it is straightforward to check that $(*)$ holds.

If $A(c)$ fails then for at least $h-k$ elements $x \in H$, $\delta_H(x) \geq c(m+1)$. In particular, if h is even then we choose $c := 2\varepsilon/h$ and obtain that $\mathrm{spread}(H) \geq (h-k)c = \varepsilon$ with probability greater than $1 - (15\varepsilon)^{k-l} \geq 1 - (15\varepsilon)^{h/4}$. If h is odd then we choose $c := 2\varepsilon/(h+1)$ and obtain that $\mathrm{spread}(H) \geq (h-k)c = \varepsilon$ with probability greater than $1 - (15\varepsilon h/(h+1))^{k-l} \geq 1 - (15\varepsilon h/(h+1))^{(h-1)/4} > 1 - (15\varepsilon)^{h/4}$. (The last inequality is the only point in the proof where we used the lower bound $\varepsilon \geq 1/30$.) □

Now we are ready to prove Theorem 1.4.

Theorem 3.3. *Let $1/30 \leq \varepsilon < 1/15$, $C > 0$, $r > (4 + 4(C+1)\log m) / \log(1/(15\varepsilon))$, and $P = \{p_1, ..., p_r\}$ a set of r random permutations of $\{1, 2, ..., m\}$. Then P is an (ε, m)-spreader with probability at least $1 - m^{-C}$.*

Proof. Let H be a nonempty subset of $\{1, 2, ..., m\}$ and $|H| = h$. By Lemma 3.2, the probability that spread$(H) < \varepsilon$ for all permutations in P is at most $(15\varepsilon)^{hr/4}$. Hence the probabity that not all nonempty subsets are ε-spread is at most

$$\sum_{h=1}^{m} \binom{m}{h} (15\varepsilon)^{hr/4} = \left(1 + (15\varepsilon)^{r/4}\right)^m - 1 < 2(15\varepsilon)^{r/4} m < m^{-C},$$

where the last inequality is equivalent to the condition imposed on r. □

3.2 Random prefixes and fast normal closure

Definition 3.4. *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 chosen uniformly from 1 through k.*

Random prefixes provide an alternative to random subproducts. Given $G = \langle S \rangle$ and an $(\varepsilon, |S|)$ spreader P, $|P| = r$, the elements of P define r orderings of S. We say that g is a *random prefix of S obtained from P* if g is a random prefix of S in the ordering defined by a randomly chosen element of P.

Lemma 3.5. *Let $G = \langle S \rangle$, P an $(\varepsilon, |S|)$ spreader, $|P| = r$, and $K \leq G$, $K \neq G$. Moreover, let g be a random prefix of S obtained from P. Then* Prob$(g \notin K) \geq \varepsilon/(2r)$.

Proof. Let $H := \{s \in S : s \notin K\}$ and let $a_1 < a_2 < ... < a_h$ denote the positions corresponding to the elements of H in the randomly chosen permutation $p \in P$. We also define $a_0 := 0$ and $a_{h+1} := |S| + 1$ and let g_j denote the random prefix of S corresponding to the first j elements of p. Then, for a fixed $1 \leq i \leq h$, either $g_j \notin K$ for all $a_{i-1} \leq j < a_i$ or $g_j \notin K$ for all $a_i \leq j < a_{i+1}$ (it is also possible that both of these events occur). So Prob$(g \notin K) \geq$ spread$(\{a_1, ..., a_h\})/2$ and, since P is an $(\varepsilon, |S|)$-spreader, spread$(\{a_1, ..., a_h\}) \geq \varepsilon$ with probability at least $1/r$. □

Lemma 3.5 provides a method to reduce the number of generators of G with roughly (up to a logarithmic factor) the same efficiency as Theorem 2.5. Given a bound L on the length of subgroup chains of G, a combination of Lemma 3.5 and Corollary 2.2 immediately gives that cLr/ε random prefixes of S obtained from P generate G with exponentially small error.

Our main goal in this section is the fast generation of normal closures. This application is based on the following lemma.

Lemma 3.6. *Suppose that $H \leq G$ but H is not a normal subgroup of G. Let $G = \langle S_G \rangle$, $H = \langle S_H \rangle$, P an $(\varepsilon, |S_G|)$ spreader, $|P| = r_1$, and Q an $(\varepsilon, |S_H|)$ spreader, $|Q| = r_2$. Moreover, let g and h be random prefixes of S_G and S_H, respectively, obtained from P and Q. Then* Prob$(h^g \notin H) \geq \varepsilon^2/(4r_1r_2)$.

Proof. By hypothesis, $K := \{k \in G : H^k = H\} \neq G$. Hence, by Lemma 3.5, Prob$(g \notin K) \geq \varepsilon/(2r_1)$. If $g \notin K$ then $X := H^{g^{-1}} \cap H \neq H$. Thus, by Lemma 3.5, Prob$(h \notin X) \geq \varepsilon/(2r_2)$. Combining the two probabilities, we obtain Prob$(h^g \notin H) \geq \varepsilon^2/(4r_1r_2)$. □

Our main result applies to black box groups where n is the length of the strings representing each group element. It also applies to permutation groups of degree n. In both cases, we know that all subgroup chains have length $O(n)$.

Theorem 3.7. *Let G and $K \leq G$ be finite groups and assume all subgroup chains in G have length $O(n)$ where n is a parameter. Assume further that G and K are given by generating sets of size $O(n)$. Then one can construct $O(n)$ generators for the normal closure of K in G with fixed but arbitrarily high probability using $O(n \log^4 n)$ group operations.*

Proof. The algorithm for constructing a sequence T of generators of the normal closure, denoted by $\langle K^G \rangle$, is described first. Fix $\varepsilon > 0$, say $\varepsilon = 1/100$, and construct an ε-spreader $P = \{\pi_1, \ldots, \pi_r\} \subset S_m$ for some large enough value of m (m will be of the order $O(n \log^2 n)$) where $r = O(\log m) = O(\log n)$. (According to Theorem 3.3, a set of random permutations will suffice.)

Let S_K and S_G be sequences of the generators for K and G. Initially set T to S_K. Pre-compute all $r(|T| + |S_G|)$ prefixes resulting from the members of P being applied to T and to S_G separately. (We use the identity element for padding, i.e. to fill slots if m is greater than S_G or the current size of T.) Since T will be augmented at each step, the prefixes corresponding to T must be stored in balanced search trees, in the following way: the elements of T are stored, in the given order, at the leaves of a balanced binary tree (including those elements to be computed in the future; their current value is the identity). Each internal node will represent the product of the elements at the leaves in the corresponding subtree. A new element of T is inserted by replacing the identity element (which currently occupies its leaf) by its actual value. Note that each insertion-update costs $O(\log m)$ group operations only. We maintain one such tree for each π_i.

Now select two members π_i and π_j of P at random. Take a random prefix τ of T^{π_i} and conjugate it by a random prefix of $S_G^{\pi_j}$.

By Lemma 3.6, if $\langle T \rangle \neq \langle K^G \rangle$ then, with probability at least $\varepsilon^2/(4r^2)$, the new conjugate φ will belong to $\langle K^G \rangle \setminus \langle T \rangle$.

Append φ to T and update each of the r trees. (One would like to add the conjugate to T only if it is not already in $\langle T \rangle$, but this cannot be efficiently tested.)

Stop when Cnr^2 conjugates have been added to T. Finally reduce the number of generators of $\langle T \rangle$ to $O(n)$.

The usual application of Corollary 2.2 shows correctness. The timing is dominated by the time it takes to update the dynamic trees. There will be $O(nr^2)$ updates, each at the cost of $O(\log n)$ group operations on each of r trees. Noting that $r = O(\log n)$, we obtain the stated time bound. □

4. Point Stabilizer Chains in Monte Carlo $O(n^3 \log^4 n)$

4.1. The result

Finally, enough machinery has been developed to present the faster algorithm for constructing point stabilizer sequences, and solving the group membership problem.

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

The result, a strong generating set U, is returned as a sequence corresponding to a chain of subgroups $G = G_0 \geq G_1 \geq \ldots \geq G_k = 1$ of length $k \leq 2n$ such that $\cup_{j=i}^{k} U[j]$ is a set of generators for G_i.

It is possible that U is not a strong generating set of G for any subgroup chain consisting of pointwise set stabilizers in a permutation action of G; in any case, no such action will be evident from the algorithm. Each member G_i of the chain will act on its own domain Ω_i, where $\Omega = \Omega_1 \subseteq \cdots \subseteq \Omega_k$ and $|\Omega_i| \leq 2n$.

However, U will enable us to generate uniformly distributed random elements of G in $O(n^2)$ time per element and so, by a method of Cooperman, Finkelstein, Sarawagi [CFS], we can construct

an SGS V for the pointwise stabilizer subgroup chain on the original permutation domain Ω in $O(n \log^2 |G|) = O(n^3 \log^2 n)$ time, justifying the title of this section. This new SGS supports membership testing in $O(n^2 \log n)$ time and requires $O(n^2 \log n)$ space. Furthermore, in $O(n^2 \log n)$ time, it is possible to construct a labelled branching from V on Ω, supporting membership testing in $O(n^2)$ time and requiring $O(n^2)$ space by Cooperman and Finkelstein [CF1].

In contrast to the previously fastest published algorithm [BLS2], this new algorithm does not require results from the classification of finite simple groups. The algorithm does run faster if appeal is made to the result from the classification that only S_n and A_n are 6-transitive, and so the algorithm is presented in a form which takes advantage of that result. An old result [Wi] by Wielandt shows that among permutation groups of degree n, only S_n and A_n are $3 \log n$-transitive. A weaker result by Jordan [Jo] states that only S_n and A_n are t-transitive where $t = c \log^2 n / \log \log n$. (The algorithmic foundation for the routine `Fast_Giant` below is provided by a simple proof of the last result by Babai and Seress [BS].) Using one of these weaker results on transitivity adds a factor of $\log n$ and $\log^2 n$, respectively, to the running time.

4.2. A lemma on primitive groups

First we recall certain concepts from permutation groups, which form the basis of a divide-and-conquer strategy. The *point stabilizer subgroup* (defined in the first section) stabilizing x is denoted G_x. The *pointwise set stabilizer* subgroup containing all group elements fixing $\{\alpha_1, \ldots, \alpha_i\}$ is denoted $G_{\{\alpha_1,\ldots,\alpha_i\}}$. For G acting on Ω, the *orbit* $\mathcal{O} \subseteq \Omega$ of $x \in \Omega$ under G is denoted $x^G = \{x^g \colon g \in G\}$. A *trivial orbit* consists of a single point. G is *transitive* on Ω (or just transitive, where the action is clear) if Ω contains only one orbit under G. G is *intransitive* otherwise. G is *doubly transitive* if G acts transitively on the set of $n(n-1)$ ordered pairs of distinct elements from Ω. For $\Psi \subseteq \Omega$ such that $\Psi^G = \Psi$, we denote by G^Ψ the set of permutations of G restricted to mappings on Ψ alone. If there exists a set $B \subset \Omega$ with $|B| \geq 2$ and $B \neq \Omega$ such that for each $g \in G$ either $B^g = B$ or $B^g \cap B = \emptyset$, then B is a *block of imprimitivity* of G. G is *imprimitive* if it is transitive and has a block of imprimitivity. If G is transitive and not imprimitive, then it is *primitive*. A primitive group which is not doubly transitive is called *uniprimitive*.

One of the elements of our strategy is an "anti-greedy" choice of the point to be stabilized in the current permutation domain: this point will always be selected from the smallest non-trivial orbit under the current group action. This choice is motivated by Theorem 2.11 which says that the cost of computing the stabilizer of a point is proportional to the length of its orbit.

To ensure that this strategy is indeed effective, we need to bound the total length of all orbits visited. These orbits are by no means disjoint, yet we require an $O(n \log n)$ bound on their total length (Lemma 4.4). The bound will be based on the following combinatorial lemma for primitive groups, of interest in its own right.

Lemma 4.2. *Let G be a primitive permutation group. Assume the stabilizer G_x has an orbit of length $d \geq 2$. Then every nontrivial subgroup of G_x has a non-trivial orbit of length $\leq d$.*

Proof. Let $\Delta \subset \Omega$ denote an orbit of G_x of length d. (Ω is the permutation domain.) Consider the *orbital graph* X corresponding to Δ. X is a directed graph with vertex set Ω and edge set $\{(x^g, y^g) : g \in G, y \in \Delta\}$. Note that in X, every vertex has out-degree d. Clearly G acts as automorphisms of X. Since G is primitive, X must be strongly connected (directed paths exist between any pair of vertices). Indeed, otherwise the strongly connected components would be blocks of imprimitivity; therefore they would be singletons, ruling out directed cycles. But every vertex has out-degree $d > 0$, therefore directed cycles exist.

If $H \leq G_x$ is nontrivial then there exists a point not fixed by H. Let us consider a directed path $x = x_0, x_1, \ldots$ connecting x to such a point. Let x_i be the first point on this path, not fixed by H ($i \geq 1$). Then all points in the orbit x_i^H are out-neighbors of x_{i-1} in X (since $x_{i-1}^H = \{x_{i-1}\}$). □

4.3 The strategy

Roughly speaking, the algorithm always tries to compute generators for the stabilizer of a point in the smallest orbit of the group at hand and add the transversal of the point stabilizer to U, the SGS to be constructed. Moreover, it tries to work in primitive groups; hence, if the action on the smallest orbit is imprimitive then it adds blocks of imprimitivity to the permutation domain, and first computes the pointwise stabilizer of these blocks. The last consideration is that if the primitive group at hand is a *giant* (symmetric or alternating group) then, counting transitivity, the algorithm recognizes it and switches to a routine designed especially to handle the giants.

The algorithm will proceed in phases. Initially we set the current group to $G_0 = G$ and the current domain to $\Omega_0 = \Omega$. Phase i begins with possibly augmenting the domain to $\Omega_i \supseteq \Omega_{i-1}$ and constructing a proper subgroup $G_i < G_{i-1}$.

As discussed above, there will be two types of phases: point-stabilizing phases and giant-elimination phases. In the former case, G_i is the stabilizer of a point in Ω_i from an orbit with primitive G_{i-1}-action. In the latter case, G_i is the pointwise stabilizer of the orbit $\mathcal{O}$ on which G_{i-1} acts as a giant.

Along the way, the algorithm constructs the set U of strong generators. Initially, $U = \emptyset$. In a point-stabilizing phase, a transversal of G_{i-1} in G_i is added to U. In a giant-elimination phase on orbit $\mathcal{O}$, an SGS for $G_{i-1}^{\mathcal{O}}$ is added to U, encoded in a labelled branching [Je] using $|\mathcal{O}| - 1$ permutations (in order to satisfy the space constraints).

Definition 4.3. A group element g is *sifted* through a subgroup chain $G = G_0 \geq G_1 \cdots \geq G_k$ with *residue* h if g can be factored as $g = hu_k \cdots u_1$ where u_i is a (right) coset representative of G_i in G_{i-1} and $h \in G_k$.

Labelled branchings allow to sift group elements, test membership, and generate random group elements as fast as complete coset tables. For completeness, we recall their definition in the case of giants. In the case of $\mathrm{Sym}(\mathcal{O})$, $\mathcal{O} = \{x_1, x_2, ..., x_m\}$, the labelled branching consists of $m-1$ permutations $p_1, p_2, ..., p_{m-1}$ such that for all $1 \leq k \leq m-1$, p_k fixes $x_1, ..., x_{k-1}$ and $x_k^{p_k} = x_{k+1}$. For $\mathrm{Alt}(\mathcal{O})$, the labelled branching contains $p_1, p_2, ..., p_{m-2}$ as above while p_{m-1} fixes $x_1, ..., x_{m-3}$ and $x_{m-2}^{p_{m-1}} = x_m$.

Since we always move to proper subgroups, the length of our subgroup chain will be less than $2n$ (Lemma 2.6).

Our fast *normal closure* routine plays a critical role in constructing the pointwise stabilizer of an orbit $\mathcal{O}$ with giant action.

We say that a set $T \subseteq K$ is a set of *normal generators* for the group K if K is the normal closure of T.

Normal generators for the pointwise stabilizer of an orbit with giant action come from two sources: the residues of sifting the generators of G_{i-1} through the stabilizer chain of $\mathcal{O}$ (accomplished via the labeled branching representation of an SGS for the giant action on $\mathcal{O}$); and the defining relations of the giant (see [BLS2, Lemma 7.2]).

4.4. The procedure

Throughout the algorithm, the points of the permutation domain are marked by integers, denoting the priority, which ones we want to stabilize first. Initially, each point has mark 0; if G is not transitive then the points of the smallest orbit get mark 1; if the action on this orbit is not primitive, a system of blocks of imprimitivity is added with mark 2; etc. Higher mark mean higher priority.

The main routine is $\texttt{SGS}(\mathtt{H}, \Psi)$ where H is a group acting on the set Ψ. In the notation of the previous subsections, H is G_i and Ψ is the corresponding domain Ω_i with the fixed points deleted.

The output is a strong generating set. Descriptions of the subsidiary routines are deferred for later.

Given a group G acting on Ω, one initially sets the global variable n to $|\Omega|$, $U = \emptyset$, and `transitivity` to 0. Groups always will be given by $O(n)$ generators (by Theorem 2.5, we may suppose that G is given this way). We set the mark $m(x) = 0$ for all $x \in \Omega$. `SGS`(G, Ω) is then called. All internal calculations on permutations are carried out on all n points and possibly additional points (such as points corresponding to the action on blocks).

Procedure `SGS`
Input: (H, Ψ) where H is a group acting on Ψ.
Global variables: n, `transitivity`.
Output: strong generating set in the form of the sequence U described above.
If $\Psi = \emptyset$ then STOP.
`mark` $\leftarrow$ `max`$\{m(x) : x \in \Psi\}$
$\mathcal{O} \leftarrow \{x \in \Psi : m(x) =$ `mark`$\}$
CASE:
$H^{\mathcal{O}}$ is intransitive:
`transitivity` $\leftarrow 0$
Let $\mathcal{O}_1 \leftarrow$ smallest orbit of $H^{\mathcal{O}}$
For $x \in \mathcal{O}_1$, $m(x) \leftarrow$ `mark` $+ 1$
Call `SGS`(H, Ψ)
$H^{\mathcal{O}}$ is imprimitive:
Let $B \leftarrow$ `Blocks`$(H, \mathcal{O})$
$\Psi \leftarrow B \cup \Psi$
For $x \in B$, $m(x) \leftarrow$ `mark` $+ 1$
Call `SGS`(H, Ψ)
Else: [$H^{\mathcal{O}}$ is primitive]
`transitivity` $=$ `transitivity` $+ 1$
If `transitivity` ≥ 6 then
[$H^{\mathcal{O}}$ is $Sym(\mathcal{O})$ or $Alt(\mathcal{O})$]
Call `Fast_Giant`$(H, \mathcal{O}, \Psi)$
[It adds an SGS of $H^{\mathcal{O}}$ to U]
$T \leftarrow$ `Normal_Generators`$(H, \mathcal{O}, \Psi \setminus \mathcal{O})$
$H \leftarrow$ `Normal_Closure`$(T, H, \Psi \setminus \mathcal{O})$
Call `SGS`$(H, \Psi \setminus \mathcal{O})$
Else
$x \leftarrow$ first$(\mathcal{O})$
$H_x \leftarrow$ `Point_Stab`$(H, x, \mathcal{O}, \Psi)$
$U \leftarrow U \cup$ (transversal of H_x in H)
Call `SGS`$(H_x, \Psi \setminus \{x\})$

The subsidiary routines are now described and timings given. With the descriptions of the subsidiary routines, it will be clear that the algorithm is correct. Descriptions of the routines are given first, followed by a proof of timing for Theorem 4.1.

In previous sections, timings were first calculated in terms of the number of group operations; the actual timing (in terms of pointer manipulations) being n-times the result. Here we deviate from this method and report directly the number of pointer manipulations, in order to make the accounting compatible with the cost of the graph theoretic subroutines.

Recall that n refers to the the full set of points on which G originally acts, while we may consider the action of a subgroup H (or its homomorphic image) acting non-trivially on a set Ψ. In

all cases, we assume that H is defined by $O(n)$ generators $\mathrm{Gen}(H)$. Further, we can always assume that $|\Psi| \leq 2n$, since adding the action on blocks of imprimitivity is the only way in which Ψ can grow. Doing so, adds a new orbit with at most half as many points as the original orbit on which the group was acting.

`Blocks`$(H, \mathcal{O})$: Returns blocks of imprimitivity for H acting on $\mathcal{O}$. This assumes that H is transitive on $\mathcal{O}$. It can be done in time $O(|\mathrm{Gen}(H)||\mathcal{O}|^2) \leq O(n|\mathcal{O}|^2)$ [At].

`Point_Stab`$(H, x, \mathcal{O}, \Psi)$: This returns $O(n)$ generators for H_x in $O(|\mathcal{O}|n^2 \log n)$ Monte Carlo time, by appeal to Theorem 2.11. In that process, a transversal for H_x in H is also constructed.

`Fast_Giant`$(H, \mathcal{O}, \Psi)$: If the restriction of H to act on $\mathcal{O}$, $H^{\mathcal{O}}$, is either $Sym(\mathcal{O})$ or $Alt(\mathcal{O})$ then the routine adds a set of size $O(|\mathcal{O}|)$ to U whose restriction to $\mathcal{O}$ is a strong generating set for $H^{\mathcal{O}}$. Given $O(n)$ generators of H, one can find $O(\log|\mathcal{O}|)$ group elements of H which generate $H^{\mathcal{O}}$ in their action on $\mathcal{O}$ in time $O(n^2 \log(|\mathcal{O}|)$, by a technique using random subproducts described in [BLS2 (section 6.2)]. Then $O(|\mathcal{O}| \log|\mathcal{O}|)$ Schreier generators for a point stabilizer subgroup of $H^{\mathcal{O}}$ can be constructed in time $O(n|\mathcal{O}| \log|\mathcal{O}|)$. Since the point stabilizer subgroup of $H^{\mathcal{O}}$ must also be symmetric or alternating, continuing the same method one can alternately find reduced generating sets of size $O(\log|\mathcal{O}|)$ and point stabilizers. Altogether, an SGS for $H^{\mathcal{O}}$ is constructed in $O(n|\mathcal{O}|^2 \log^2|\mathcal{O}|)$ time such that each point stabilizer is generated by $O(\log|\mathcal{O}|)$ elements. In $O(n|\mathcal{O}|^2)$ further time, the two generators of $H^{\mathcal{O}}$ needed for the presentation of $H^{\mathcal{O}}$ in [BLS2(section 7)] and a labelled branching constructed from these two generators can be obtained. We add this labelled branching to U. Note that since $|\mathcal{O}| = O(n)$, the total time requirement of this subroutine is $O(n^2|\mathcal{O}| \log^2|\mathcal{O}|)$.

`Normal_Generators`$(H, \mathcal{O}, \Psi - \mathcal{O})$: Group elements T (called *normal generators*) are constructed whose action on $\mathcal{O}$ is trivial, such that $\langle T^H \rangle = H^{\Psi - \mathcal{O}}$. For a general description, see [BLS2(Lemma 7.1)]. In this setting, the routine is invoked only when $\mathcal{O}$ is an orbit on which H acts as the symmetric or alternating group. Having constructed a point stabilizer chain for the points of $\mathcal{O}$, all generators of H are sifted through the point stabilizer chain in time $O(n^2|\mathcal{O}|)$, until their residue acts trivially on $\mathcal{O}$. The residues are added to the set T. Then, a presentation for $H^{\mathcal{O}}$ of size $|\mathcal{O}|$ is added to T and the conjugates $\{h^g : g \in \mathrm{Gen}(H), h \in \mathrm{Gen}(H^{\mathcal{O}})\}$ are sifted as in [BLS2]. (Note that $H^{\mathcal{O}}$ has only two generators.) Thus the overall timing to create the normal generators will be $O(n^2|\mathcal{O}|)$.

`Normal_Closure`$(T, H, \Psi \setminus \mathcal{O})$: The normal closure can be executed in time $O(n^2 \log^4 n)$ by Theorem 3.1.

4.5 Timing

We estimate the total time requirement of calls to these subroutines during the execution of the algorithm.

First we observe that $|\Omega_i| \leq 2n$ for all i. Indeed, Ω forms the set of leaves of a forest defined in an obvious manner with nodes corresponding to certain blocks of imprimitivity. All auxiliary points added to Ω are internal nodes of this forest with at least two children each.

Next we claim that $\sum|\mathcal{O}| \leq 2n$, where $\mathcal{O}$ runs over orbits of intermediate groups H such that `Fast_Giant`$(H, \mathcal{O}, \Psi)$ is called. This is true since since these orbits are disjoint. We obtain immediately that calls to `Fast_Giant` add $O(n)$ permutations to U and the total time requirement for calls of `Fast_Giant`, `Normal_Generators`, `Normal_Closure` is $O(n^3 \log^4 n)$.

To estimate the total cost, we need the following lemma.

Lemma 4.4. *The total length of all orbits selected during the procedure is* $O(n \log n)$.

Proof. As we have seen in the previous paragraph, the total length of orbits in calls of `Fast_Giant` is $O(n)$. Hence, it is enough to estimate the total length of orbits in calls of `Point_Stab`. This is where Lemma 4.2 will be required.

Each recursive call of `Point_Stab` has its "current orbit" $\mathcal{O}$; let us call the points of $\mathcal{O}$ *active* in that phase. We claim that each point is active in $O(\log n)$ phases only. Since the total number of points is $\leq 2n$, the conclusion follows by counting the incidences of current orbits and their (active) points in two ways.

Let x be a point (original or auxiliary). Let $\mathcal{O}_1 \supset \mathcal{O}_2 \supset \cdots \supset \mathcal{O}_r$ be the sequence of current orbits containing x and let $n_i = |\mathcal{O}_i|$.

We claim that either $n_{i+1} = n_i - 1$ or $n_{i+1} \leq n_i/2$, and that the former cannot occur more than five times in a row. Clearly, this implies the claimed $O(\log n)$ bound.

The former case (minimum descent) occurs exactly when the action on the action on the current orbit is doubly transitive and its stabilizer is primitive. If this continues more than five times in a row, the orbit experiences 6-transitive action, hence a giant, and the next recursive call terminates the orbit.

The action on the current orbit is always primitive. Suppose now that it is *not* doubly transitive. Then the stabilizer is intransitive, therefore at least one of its orbits has size $d \leq (n_i - 1)/2$. But then, by Lemma 4.2, whenever x becomes active again, the smallest orbit has size $\leq d$, therefore $n_{i+1} < n_i/2$.

Suppose now that the current orbit is doubly transitive but its stabilizer is imprimitive. Then we switch to a new orbit consisiting of auxiliary points representing blocks of imprimitivity of $\mathcal{O}_n$, and we shall have fixed all of those points before making x active again. This means $\mathcal{O}_{n+1}$ is included in a block of imprimitivity of $\mathcal{O}_n$, hence $n_{i+1} \leq n_i/2$. □

Lemma 4.4 and Theorem 2.11 immediately imply that the total cost of calls of `Point_Stab` is $O(\sum n^2|\mathcal{O}| \log n) = O(n^3 \log^2 n)$. Lemma 4.4 also implies that the number of transversal elements added to U is $O(n \log n)$. As a third application of Lemma 4.4, we observe that the total cost of calls of `Blocks` is $O(\sum n|\mathcal{O}|^2) = O(n^3 \log n)$. This concludes the proof of Theorem 4.1.

5. Open problem

The following problem, mentioned in Section 1.3, is of considerable interest because it affects the reliability of our most efficient algorithms.

Problem. Construct an *explicit* set P_m of $O(\log m)$ permutations of a set of m elements such that P_m is a (c, m)-spreader for some constant $c > 0$.

(See Section 1.3 for the definition of spreader.)

References.

[At] M.D. Atkinson, "An algorithm for finding the blocks of a permutation group", *Math. Comp.* **29** (1975), pp. 911–913.

[Ba1] L. Babai, "Monte Carlo algorithms in graph isomorphism testing", Université de Montréal Tech. Report D.M.S. 79-10, 1979.

[Ba2] L. Babai, "On the length of chains of subgroups in the symmetric group", *Comm. in Algebra* **14** (1986), pp. 1729–1736.

[Ba3] L. Babai, "Local expansion of vertex-transitive graphs and random generation in finite groups", Proc. 23rd ACM STOC, 1991, pp. 164–174.

[BBR] L. Babai, R. Beals, and D. Rockmore, "Deciding finiteness of matrix groups in deterministic polynomial time", Proc. ISSAC'93 (Internat. Symp. on Symbolic and Algebraic Computation), Kiev 1993, ACM Press 1993, pp.117–126.

[BCFS] 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, pp. 200–209, 1991.

[BLS1] L. Babai, E. Luks, and Á. Seress, "Permutation groups in NC", *Proc. 19th ACM STOC* (1987), pp. 409–420.

[BLS2] L. Babai, E. Luks, and Á. Seress, "Fast management of permutation groups", *Proc. 28th IEEE FOCS* (1988), pp. 272–282.

[BLS3] L. Babai, E. M. Luks, and Á. Seress, "Computing composition series in primitive groups", In: "Groups and Computation", (L. Finkelstein and W. M. Kantor, eds.), DIMACS Ser. in Discr. Math. and Theor. Comp. Sci. Vol 11, A.M.S. 1993, pp. 1–16.

[BLS4] L. Babai, E. Luks, and Á. Seress, "Fast Deterministic Management of Permutation Groups", in preparation.

[BS] L. Babai and Á. Seress, "On the degree of transitivity of permutation groups: a short proof", *J. Combinatorial Theory: Series A* **45**(2) (1987), pp. 310–315.

[BSz] L. Babai and E. Szemerédi, "On the complexity of matrix group problems I", *25th IEEE FOCS*, Palm Beach FL (1984), pp. 229–240.

[BeB] R. Beals and L. Babai, "Las Vegas algorithms for matrix groups", Proc. 25th ACM STOC, 1993, Palo Alto CA, pp. 427–436.

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

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

[CF1] G. Cooperman and L. Finkelstein, "A strong generating test and short presentations for permutation groups", *J. Symbolic Computation* **12** (1991), pp. 475–497.

[CF2] G. Cooperman and L. Finkelstein, "Combinatorial tools for computational group theory", *Proceedings of DIMACS Workshop on Groups and Computation* (L. Finkelstein and W.M. Kantor, eds.), DIMACS-AMS **11**, AMS, Providence, RI. 1993, pp. 53–86,

[CFS] G. Cooperman and L. Finkelstein, "A randomized base change algorithm for permutation groups", *J. Symbolic Computation*, to appear. (See preliminary version: G. Cooperman, L. Finkelstein, and N. Sarawagi, "A random base change algorithm for permutation groups", *Proc. of the 1990 Internat. Symp. on Symbolic and Algebraic Manipulation* (ISSAC-90), ACM Press and Addison-Wesley, 1990.)

[FHL] M. Furst, J. Hopcroft, and E. Luks, "Polynomial time algorithms for permutation groups", *Proc. 21st IEEE FOCS* (1980), pp. 36–41.

[Ha] M. Hall, Jr., *The Theory of Groups*, Macmillan, New York, 1959.

[Je] M. Jerrum, "A compact representation for permutation groups", *J. Algorithms* **7** (1986), pp. 60–78.

[Joh] D. S. Johnson, "The NP-completeness column", *J. Algorithms* **5** (1984), pp. 433–447.

[Jo] C. Jordan, "Nouvelles recherches sur la limite de transitivité des groupes non alternés", *Bull. Soc. Math. France* **1** (1873), pp. 35–60.

[Kn] D.E. Knuth, "Notes on efficient representation of perm groups" *Combinatorica* **11** (1991), pp. 57–68 (preliminary version circulated since 1981).

[Lu] E. M. Luks, "Computing in solvable matrix groups", In: 33rd IEEE FOCS, IEEE Comp. Soc. 1992, pp. 111–120.

[Si] C.C. Sims, "Computation with permutation groups", in *Proc. Second Symposium on Symbolic and Algebraic Manipulation*, edited by S.R. Petrick, ACM, New York, 1971.

[Wi] H. Wielandt, "Abschätzungen für den Grad einer Permutationsgruppe von vorgeschriebenem Transitivitätsgrad", Dissertation, Berlin, 1934. *Schriften Math. Seminars Inst. Angew. Math. Univ. Berlin* **2** (1934), pp. 151–174.

`file cooperman.dir/stoc91.dir/journal.tex`