J. Symbolic Computation (2003) **17**, 513–528

A Random Base Change Algorithm for Permutation Groups

GENE COOPERMAN† AND LARRY FINKELSTEIN†

College of Computer Science

Northeastern University

Boston, Mass. 02115

U.S.A.

(Received 2 April, 1991
Revised 16 December, 1993)

A new random base change algorithm is presented for a permutation group G acting on n points whose worst case asymptotic running time is better for groups with a small to moderate size base than any known deterministic algorithm. To achieve this time bound, the algorithm requires a random generator $Rand(G)$ producing a random element of G with the uniform distribution and so that the time for each call to $Rand(G)$ is bounded by some function $f(n, G)$. The random base change algorithm has probability $1 - 1/|G|$ of completing in time $O(f(n, G) \log |G|)$ and outputting a data structure for representing the point stabilizer sequence relative to the new ordering. The algorithm requires $O(n \log |G|)$ space and the data structure produced can be used to test group membership in time $O(n \log |G|)$. Since the output of this algorithm is a data structure allowing generation of random group elements in time $O(n \log |G|)$, repeated application of the random base change algorithms for different orderings of the permutation domain of G will always run in time $O(n \log^2 |G|)$. An earlier version of this work appeared in Cooperman and Finkelstein (1992b).

1. Introduction

Let G be a permutation group acting on an n-element set Ω with G specified by a generating set S. 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

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

where $G^{(i)} = G_{\alpha_1,\ldots,\alpha_{i-1}}$, $1 \leq i \leq n$. The set 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 work of these authors was supported in part by the National Science Foundation under grants number CCR-8903952 and CCR-9204469.

0747–7171/90/000000 + 00 $03.00/0

The *group membership problem* is to decide whether a given element $g \in \mathrm{Sym}(\Omega)$ is also a member of G. Once one has a strong generating set, one can efficiently decide the group membership problem by straightforward techniques. A set $B \subseteq \Omega$ is called a *base* for G if for any non-identity element $g \in G$ there is a $x \in B$ such that $x^g \neq x$. The set $\{\alpha_i\colon |\alpha_i^{(G^{(i)})}| > 1\}$ is called a *base* for G relative to the ordering α, and each α_i in the set is called a *base point*.

Most important algorithms for performing computations with G assume the knowledge of a strong generating set and base relative to some ordering of Ω. Equally important is the computation of a strong generating set and base for G relative to α, given a strong generating set and base for G relative to some initial ordering α'. This is called a *base change algorithm* and finds applications not only in algorithms designed for discovering structural properties of permutations groups (Cooperman, Finkelstein and Luks, 1989; Sims, 1971), but also in the application of computational group theory to search problems (Brown, Finkelstein and Purdom, 1988; Butler and Lam, 1985; Cooperman and Finkelstein, 1992a; Lam and Thiel, 1989; Leon, 1984). In this class of problems, base changes are used repeatedly, and the availability of a very fast base change algorithm is crucial in order to maximize the use of group theoretic methods to reduce the size of the search space. The base change algorithm of this paper reduces the asymptotic bounds on the expected space and time requirements for groups with a small base.

The first algorithm for performing a change of base was given by Sims (1971). This algorithm employs a clever trick and uses Sims's Schreier vector data structure for representing the point stabilizer sequence. Later, Brown, Finkelstein and Purdom (1989) were able to generalize Sims's trick in combination with a new data structure for representing the point stabilizer sequence. This improved the running time of Sims's algorithm by two orders of magnitude. The new data structure, a labelled branching, was first described by Jerrum (1986). This paper is distinguished from earlier efforts by using random methods. An important forerunner of this approach is the work of Babai (1979) of ten years earlier, in which he used random group elements in the context of graph isomorphism to construct a strong generating set relative to a non-traditional subgroup chain.

A later refinement (Cooperman, Finkelstein and Luks, 1989) of the deterministic algorithm (Brown, Finkelstein and Purdom, 1989) improves the running time to $O(bn^2)$ where b is the size of the base for G relative to α. That base change algorithm reduces the worst case running time and works well with groups having a moderate to large size base. Unfortunately, because a labelled branching data structure requires $\Theta(n^2)$ storage for transitive groups, the best case running time for this algorithm is $\Omega(n^2)$, even when the group has a very small base.

To overcome this lower bound on space, the Schreier vector data structure is commonly used for representing the point stabilizer sequence, and tends to be more space efficient than a labelled branching, in practice, when b is small. One can view the Schreier vector data structure relative to α as a sequence of directed trees, denoted by $\mathcal{S}(i)$, one for each base point α_i, with the tree rooted at i, and edges of the tree labelled by elements of $G^{(i)}$. $\mathcal{S}(i)$ stores all the information required to compute both $\alpha_i^{(G^{(i)})}$ and for each $\beta \in \alpha_i^{(G^{(i)})}$ an element of $G^{(i)}$ which move α_i to β. If d_i denotes the depth of $\mathcal{S}(i)$, then in the worst case, recovering an element $G^{(i)}$ which moves α_i to a given point in $\alpha_i^{(G^{(i)})}$ may require as many as $d_i - 1$ permutation multiplications, or $O(n^2)$ time. This implies that membership testing can take time $O(bn^2)$ in the worst case.

This paper presents a restricted form of the Schreier vector data structure which requires $O(n \log |G|)$ space and such that membership testing takes $O(n \log |G|)$ time. This makes possible a random base change algorithm with worst case running time $O(n \log^2 |G|)$, which is provably faster than the deterministic algorithms for groups with a small base. The key idea is to use random elements of G to build the restricted Schreier vector data structure. On average the depth of each Schreier tree $\mathcal{S}(i)$ is proportional to $\log(|\alpha_i^{(G^{(i)})}|)$. This, of course, can always be achieved by merely throwing in extra generators. What is surprising is that we are in fact able to build the data structure so that it takes space proportional to $n \log |G|$.

In order to state our result, we will assume that $\mathit{Rand}(G)$ is a random generator for G which generates random elements according to the uniform distribution. We define a *short* Schreier vector data structure for G relative to an ordering to be one which requires space for at most $20n \log_2 |G|$ permutations and such that membership testing using the data structure requires at most $20n \log_2 |G|$ permutation multiplications and inverses. (The choice of 20 is determined by calculations in the body of the paper.)

THEOREM A. *Let α be a fixed ordering of Ω. A random base change algorithm can be described which has probability $1 - 1/|G|$ of constructing a short Schreier vector data structure for G relative to α after processing at most $20 \log_2 |G|$ random elements of G.*

The proof of Theorem A involves a new technique for building a short Schreier vector data structure for G relative to α. The Schreier vector data structure created by our algorithm is slightly more wasteful of space than one created from a strong generating set without any redundancies. It has been previously shown how to eliminate the redundancies in time $O(n|S| + \log n)$ (Cooperman and Finkelstein, 1992b). Since this work was done, we have also generalized the techniques to demonstrate a $O(nb \log^2 n)$ time algorithm in the important special case when the base change permutation is a single cycle, for b the size of a base (Cooperman and Finkelstein, 1992c).

THEOREM B. *If each invocation of $\mathit{Rand}(G)$ requires $O(n \log |G|)$ time, then the probability is $1 - 1/|G|$ that the random base change algorithm will construct a short Schreier vector data structure in $O(n \log^2 |G|)$ time. For the general case in which $\mathit{Rand}(G)$ requires $O(f(n, G))$ time, a short Schreier vector data structure is constructed in $O(f(n, G) \log |G|)$ time with probability $1 - 1/|G|$.*

$\mathit{Rand}(G)$ is typically constructed as follows. Let

$$G = G_1 \supseteq G_2 \supseteq \cdots \supseteq G_b = \{e\}$$

be a chain of subgroups, and let T_i be a transversal for G_{i+1} in G_i with $1 \leq i \leq b - 1$. (Normally, it is assumed that the identity belongs to each transversal.) Each invocation of $\mathit{Rand}(G)$ produces an element $g = g_{b-1} g_{b-2} \cdots g_1$ where g_i is randomly chosen from the set T_i, $1 \leq i \leq b - 1$. Given this construction, it easy to see that $\mathit{Rand}(G)$ has the uniform distribution.

In a common situation, the point stabilizer sequence for G relative to an initial ordering α' is known and this sequence can be used to construct $\mathit{Rand}(G)$. The time complexity for $\mathit{Rand}(G)$ depends on the nature of the data structure used to represent the transversals T_i, $1 \leq i \leq b - 1$. The algorithms of Knuth (1991), Jerrum (1986) and the output of the base change of this paper all allow computation of $\mathit{Rand}(G)$ in time $O(n \log |G|)$. Sims's algorithm (1971) creates a Schreier vector data structure which may require up to n^2 time for generation of random group elements. However, it has often been observed that Sims's algorithm tends to produce "shallow" Schreier vectors in practice, allowing computation of $Rand(G)$ in time $O(n \log |G|)$. Typically, the random base change algorithm is used to obtain the point stabilizer sequence for G relative to a new ordering α of the underlying point set, but this does not have to be the case.

Once $Rand(G)$ has been defined, the random base change algorithm can be applied to obtain the point stabilizer sequence for any permutation representation of G. This idea was first exploited by Babai (1979), who is also responsible for one of the key results (see Lemma 4.1). Thus a major innovation of this paper is the fact that our random base change algorithm constructs a short Schreier vector data structure. This is responsible for the overall improvements in both time and space as described in Theorems A and B.

The random base change algorithm of this paper has more general applicability than previous base change algorithms, since it requires only a source of random group elements, $Rand(G)$, and not an initial strong generating set. This feature is used to advantage in a recent Monte Carlo algorithm to construct a point stabilizer sequence in time $O(n^3 \log^c n)$ (Babai, Cooperman, Finkelstein, Luks and Seress, 1991). Initially, the Monte Carlo algorithm, and related deterministic versions (Babai, Luks and Seress, 1988) construct subgroup chains not corresponding to any point stabilizer sequence, but serving only as an intermediate step toward constructing strong generating sets. In that situation, a single application of a traditional base change would not allow one to recover a strong generating set for any given ordering. However, the intermediate subgroup chains do allow time-efficient generation of uniformly random group elements, and thus implement the random function $Rand(G)$. The random base change algorithm of this paper then efficiently converts the intermediate chain to a more traditional point stabilizer sequence with respect to the given ordering.

The significance of Theorem B may also be seen in the context of a search problem where one already has computed a strong generating set S for G relative to some ordering α' and must repeatedly compute the point stabilizer sequence relative to a new ordering α. If a labelled branching has been used to construct S, then each invocation of $Rand(G)$ can be performed in time $O(n \log |G|)$ and so Theorem B applies. On the other hand, if a Schreier vector data structure has been used, then there is no guarantee that an invocation of $Rand(G)$ can be performed in time $O(n \log |G|)$. In this case, one can simply invoke Theorem A to build a short Schreier vector data structure relative to α', which will then be used in further applications. Otherwise, if n is small enough, then one can use a fast algorithm (Cooperman and Finkelstein, 1992b) to build a labelled branching for G relative to α' in time $O(n|S| + n^2)$ and then use this data structure for future computation of $Rand(G)$. In any event, the time to build a data structure for computing $Rand(G)$ so that each invocation takes time $O(n \log |G|)$ is dominated by the original construction of a strong generating set for G relative to α'.

2. Group Membership Data Structures

In this section, we present the relevant background material on data structures used to represent the point stabilizer sequence for G relative to some ordering α.

Let S be a generating set for G. For each i, $1 \le i \le n-1$, let $U^{(i)}$ be a set of elements of $\langle S \cap G^{(i)} \rangle$ such that $U^{(i)}$ is in 1-1 correspondence with the elements of $\alpha_i^{\langle S \cap G^{(i)} \rangle}$ under the map

$$u \in U^{(i)} \mapsto \alpha_i^u.$$

Assume that the identity element belongs to $U^{(i)}$. Set $U = \cup_{i=1}^{n-1} U^{(i)}$. We say that U is *fully augmented for* S with respect to α, since U incorporates all the information for the cosets of the point stabilizer sequence implicit in S. Note that S is a strong generating set for G relative to α if and only if $|U^{(i)}| = [G^{(i)} : G^{(i+1)}]$ for each base point α_i.

We say that the permutation g *factors through* U if we can write g in the form

$$g = g_{i_b} g_{i_{b-1}} \cdots g_{i_1},$$

where $\{\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_b}\}$ is the base for G relative to α with $i_1 < i_2 < \ldots < i_b$ and $g_{i_j} \in U^{(i_j)}$, $1 \le j \le b$. In this case, the factorization is clearly unique. It is straightforward to construct an algorithm which factors g in this form. Clearly, if g can be factored in this form, then $g \in G$. The converse, that every $g \in G$ can be factored in this form, is true if and only if S is a strong generating set. In fact, when S is a strong generating set, the standard membership test for g returns true if and only if it succeeds in factoring g in this form.

In order to save space, the elements of U typically are not stored explicitly. The two data structures we will use in this paper for storing U implicitly are the Schreier vector data structure and labelled branchings.

One can consider a Schreier vector data structure $\mathcal{S}$ relative to α as a sequence of directed trees denoted by $\mathcal{S}(i)$ rooted at i, one for each base point α_i. The nodes of $\mathcal{S}(i)$ are the indices j such that α_j is in the orbit ${\alpha_i}^{\langle S \cap G^{(i)} \rangle}$ and the edges are of the form (j, k) with label $s \in S \cap G^{(i)}$ and such that ${\alpha_j}^s = \alpha_k$. Given $j \in \mathcal{S}(i)$, one can construct an element $p \in G^{(i)}$ which moves α_i to α_j by taking the product of the edge labels along the path from i to j. In this case, $\mathcal{S}(i)$ can be built from $S \cap G^{(i)}$ using a breadth first search and the points in $\mathcal{S}(i)$ will then coincide with $\alpha_i^{\langle S \cap G^{(i)} \rangle}$. The set $U^{(i)}$ will then consist of all elements of $\langle S \cap G^{(i)} \rangle$ formed from $\mathcal{S}(i)$ in this way. Note that since the depth of $\mathcal{S}(i)$ can be as much as $n - i$, it can take as many as $n - i - 1$ multiplications to compute a single element of $U^{(i)}$. Thus testing if an element of G factors through U using the Schreier vector data structures takes time $O(bn^2)$ where b is the size of a base relative to α.

In the literature, the tree $\mathcal{S}(i)$ is usually represented by two arrays, a parent array (whose entries are the index of the parent in the directed tree, or NIL if a node has no parent) and an array known as a *Schreier vector* which specifies the element of S that is the label of the edge from the parent of a node to the node (if the parent exists). In this paper, we shall refer to such a tree representation as a *Schreier tree*, and the set of trees for i is called a *Schreier vector data structure*. A Schreier tree $\mathcal{S}(i)$ is *complete* if its nodes are $\alpha_i^{\langle G^{(i)} \rangle}$, and a Schreier vector data structure is *complete* if its component trees are complete.

An alternative data structure is the labelled branching data structure described by Jerrum (1986) (see also Cooperman and Finkelstein, 1992b). Formally, a *branching* on Ω is a directed forest with vertices $1, \ldots, n$. A branching $\mathcal{B}$ is said to be a *labelled branching for* G relative to α, if each edge (i, j) is labelled by a permutation σ_{ij} so that the following property holds:

(i) $\sigma_{ij} \in G^{(i)}$ and moves α_i to α_j.

(ii) The set of edge labels of $\mathcal{B}$ generate G.

If j is reachable from i in $\mathcal{B}$, then the product of the edge labels along the path from i to j, defines an element of $G^{(i)}$ which moves α_i to α_j. In this way, we can construct the set $U^{(i)}$ for each i and hence for $U = \cup_{i=1}^{n-1} U^{(i)}$. U is *fully augmented* with respect to S if given an arbitrary $\alpha_j \in \alpha_i^{\langle S \cap G^{(i)} \rangle}$, there is a path in $\mathcal{B}$ from i to j. From here on, U is assumed to be fully augmented. Rather than directly storing the edge labels of $\mathcal{B}$, we store node labels $\tau[i]$ for each node α_i. We define $\tau[i]$ to be the product of the edge labels along the path from r to α_i where node r is the root of the largest subtree of $\mathcal{B}$ containing α_i. (If $r = \alpha_i$ then $\tau[i]$ is the identity.) It then follows that the product of edge labels from node α_i to node α_j may be realized as the single permutation multiply $\tau[i]^{-1}\tau[j]$. In this data structure, the cost of computing a coset representative is $O(n)$, and so testing if an element of G factors through U using this data structure takes time $O(bn)$. However, a labelled branching requires $O(n^2)$ storage. Further, if G is transitive, then the cost of storing a labelled branching is $\Theta(n^2)$.

This leads us to a classic space-time tradeoff problem in the case where b is small in comparison to n and where $supp(G) = \Omega$ (i.e. G doesn't fix any points of Ω). Since it can be shown that a strong generating set S can always be chosen to have cardinality bounded by $\log_2 |G| \leq b \log_2 n$ (Cooperman and Finkelstein, 1992b), it follows that the storage required for a Schreier vector data structure is $O(n \log |G|)$ in comparison to $\Theta(n^2)$ for labelled branchings, a substantial improvement. In contrast, factoring an element for labelled branchings takes time $O(bn)$ whereas the corresponding time for a Schreier vector data structure is $O(bn^2)$. In the context where we have a random generator $Rand(G)$ for G, we will be able to show that with very high probability, we can construct a complete Schreier vector data structure for G which requires $O(n \log |G|)$ storage and where factoring an element of G takes time $O(n \log |G|)$.

3. Short Schreier Trees

In this section, we will assume that we have a random generator $Rand(G)$ for G so that each element of G has an equal likelihood of being chosen. As before, α will represent some ordering of the points of Ω. We will present a construction for a Schreier tree used to describe an orbit of G acting on Ω so that with very high probability the tree will have log depth. This will then be applied in the next section to the random base change algorithm.

In order to motivate the material to follow, suppose for simplicity that G is transitive on Ω and α is the trivial ordering. Suppose we are trying to build $\mathcal{S}(1)$ for a Schreier vector data structure $\mathcal{S}$ for G relative to α. Let P represent the the points so far included in $\mathcal{S}(1)$. If we choose g using $Rand(G)$, then for each point $p \in P$ such that $p^g \in P^g - P$, a new node in $\mathcal{S}(1)$ can be constructed with the edge (p, p^g) labelled by g. The addition of these new nodes increases the depth of $\mathcal{S}(1)$ by at most 1. Thus $\mathcal{S}(1)$ will have logarithmic depth if we can build up the tree using $O(\log n)$ randomly chosen elements in this way. Theorem 3.5 shows that this can be done with very high probability.

Let P be an arbitrary, non-empty subset of some orbit $\mathcal{O}$ of G acting on Ω. Define the random variable X_P on G by setting

$$X_P(g) = |P^g - P|.$$

LEMMA 3.1.

$$E(X_P) = (|\mathcal{O} - P|/|\mathcal{O}|)|P|.$$

PROOF. Without loss of generality, we may assume that G is transitive, and set $\mathcal{O} = \Omega$.

Now, let $\chi_{P,p}: G \mapsto \{0,1\}$ be a family of functions parameterized by P and $p \in P$, defined by $\chi_{P,p}(g) = 0$ if $p^g \in P$ and 1 otherwise. We claim that

$$\sum_{g \in G} \chi_{P,p}(g) = |G|(n - |P|)/n.$$

To prove this first note that the elements of G can be partitioned into right cosets of G_p so that all elements of each coset map p into the same point. This implies that, $\chi_{P,p}$ is constant on any coset with the value being 0 if the elements map p into P and 1 otherwise. Thus the fraction of the elements of G for which $\chi_{P,p}$ has the value 1 is $(n - |P|)/n$ and the claim follows.

We use this claim to prove our main result as follows.

$$\begin{aligned} E(X_P) &= \sum_{g \in G} X_P(g)(1/|G|) = \sum_{g \in G} \sum_{p \in P} \chi_{P,p}(g)/|G| \\ &= \sum_{p \in P} \left(\sum_{g \in G} \chi_{P,p}(g)/|G| \right) = \sum_{p \in P} ((n - |P|)/n) \\ &= |P|(n - |P|)/n \end{aligned}$$

□

LEMMA 3.2. *Let c be an arbitrary constant satisfying $|P|/|\mathcal{O}| \leq c \leq 1$. Then for random $g \in G$,*

$$\Pr(|P^g - P| > (1-c)|P|) \ \geq \ 1 - |P|/(|\mathcal{O}|c).$$

PROOF. Recall that $X_P(g) = |P^g - P|$. Let

$$x \ = \ \Pr(X_P(g) > (1-c)|P|).$$

Noting that $X_P(g) \leq |P|$ for all g,

$$E(X_P) \ \leq \ (1-x)(1-c)|P| + x|P| \ = \ (1 - c + cx)|P|.$$

Combining this with Lemma 3.1 yields

$$x \geq 1 - |P|/(|\mathcal{O}|c),$$

as required. □

LEMMA 3.3. *Let c be an arbitrary constant satisfying $|P'|/|\mathcal{O}| \leq c \leq 1$, where $P' = \mathcal{O} - P$. Then for random $g \in G$,*

$$\Pr(|P^g - P| > (1-c)|P'|) \ \geq \ 1 - |P'|/(|\mathcal{O}|c).$$

PROOF. Note that $|P^g - P| = |(P')^g - P'|$, and apply Lemma 3.2 on P' to obtain the result. □

Corollary 3.4. *For random* $g \in G$, $\Pr(|P^g - P| > (|P||P'| - \min(|P|^2, |P'|^2)/2)/|\mathcal{O}|) \geq 1/3$, *where* $P' = \mathcal{O} - P$.

Proof. There are two cases. If $|P| \leq |P'|$, set $c = (3/2)|P|/|\mathcal{O}|$, in Lemma 3.2, and note that

$$(1-c)|P| = (|P||P'| - |P|^2/2)/|\mathcal{O}|.$$

If $|P'| \leq |P|$, then set $c = (3/2)|P'|/|\mathcal{O}|$ in Lemma 3.3, and note that

$$(1-c)|P'| = (|P||P'| - |P'|^2/2)/|\mathcal{O}|.$$

□

Let us return to the original problem of constructing a Schreier tree for an orbit $\mathcal{O}$ of G. We describe a simple algorithm for building the Schreier tree. Again, for simplicity, we assume the trivial ordering α.

Random-Build-Schreier-Tree
Input: A random generator $Rand(G)$ and a point j in an orbit of G.
Output: A Schreier tree $\mathcal{T}$.
Local Variable: $P_0, P_1, \ldots$ is a sequence of sets in which P_i is the set of nodes in $\mathcal{T}$ after the i^{th} iteration.
Initialize $\mathcal{T}$ to the trivial tree rooted at j
Initialize $P_0 \leftarrow \{j\}$, $i \leftarrow 0$
While $|P_i| \neq |\mathcal{O}|$ do
  Let $g \leftarrow Rand(G)$
  For each $p \in P_i$ such that $p^g \in P_i^g - P_i$ do
    Connect p to p^g in $\mathcal{T}$ with edge label g
  Set $P_{i+1} \leftarrow$ the nodes currently in $\mathcal{T}$
  Set $i \leftarrow i + 1$
Return($\mathcal{T}$)

In the above algorithm, we have used as a stopping criteria that $|P_i| = |\mathcal{O}|$. Of greater interest for its use in section 4 is the question of how many iterations of the main loop are required before we have a high probability of stopping. Chernoff's bound (1952) is the primary tool in deriving this estimate.

Chernoff 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

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

Theorem 3.5. *There exists a constant* $c > 0$ *(one can choose* $c = 20$*) such that for all* $\delta \geq 1$, *if* $d = \lceil \delta c \log_2 |\mathcal{O}| \rceil$, *then in Random-Build-Schreier-Tree,* $P_i = \mathcal{O}$, *for some* $i \leq d$, *with probability at least* $1 - 1/|\mathcal{O}|^\delta$.

Proof. We define an iteration of the main loop of Random-Build-Schreier-Tree to be a success if the event specified in Corollary 3.4 occurs. Specifically, if $|P_i| < |\mathcal{O}|$, then the i^{th} iteration is a success if $|P_i - P_i^g| > (|P_i||P_i'| - \min(|P_i|^2, |P_i'|^2)/2)/|\mathcal{O}|$. This will happen with probability at least 1/3 by Corollary 3.4.Throughout, $g = g_i$ for (g_i) a sequence of independently chosen random group elements, and we suppress the subscript i. We first determine an upper bound on the number of consecutive successes required in order to guarantee that the algorithm completes. Since an unsuccessful iteration will never reduce the number of later successful iterations needed to guarantee completion, the upper bound will also apply to the number of non-consecutive successes. For simplicity of notation, we assume that $\mathcal{O} = \Omega$ so that $|\mathcal{O}| = n$.

The upper bound is estimated by dividing the analysis into two regimes, $\{i\colon\ i > 0, |P_{i-1}| \leq n/2$, iteration i a success$\}$ and $\{i\colon\ |P_{i-1}| > n/2$, iteration i a success$\}$, and find upper bounds m and t on the respective set sizes. First, consider the case when $|P_i| \leq n/2$ and P_i is a success. In this case,

$$|P_{i+1}| = |P_i \cup P_i^g| = |P_i| + |P_i - P_i^g| \geq |P_i|(4n - 3|P_i|)/2n$$

by Corollary 3.4. Since $|P_i| \leq n/2$, it follows that $|P_{i+1}| \geq 5/4|P_i|$. In particular, there exists an $m \leq \lceil \log_{5/4}(n/2) \rceil$, such that after m iterations we will have accounted for at least half the points.

In the second regime, let m be the first index for which $|P_m| > n/2$. Set $q_i = (n - |P_{m+i}|)/n$ for $i \geq 0$. Then

$$q_{i+1} = (n - |P_{m+i+1}|)/n = (n - (|P_{m+i}| + |P_{m+i}^g - P_{m+i}|))/n.$$

Since we are assuming that all P_j are successes (satisfy the event of Corollary 3.4), that and $|P_{m+i}| > n/2$ yield

$$|P_{m+i}^g - P_{m+i}| > (2n|P_{m+i}| - (3/2)|P_{m+i}|^2 - n^2/2)/n.$$

Substituting this in the previous equation and simplifying, we obtain

$$q_{i+1} < (3/2)q_i^2.$$

Since $q_0 < 1/2$, it follows, using repeated substitution, that

$$q_t < (3/2)^{2^t-1}(1/2)^{2^t} = (2/3)\,(3/4)^{2^t}.$$

For t sufficiently large, $q_t < 1/n$. Since nq_t is an integer, this implies $q_t = 0$. If we choose $t \geq \lceil \log_2 \log_2(2n/3) - \log_2 \log_2(4/3) \rceil$, then $q_t = 0$ and the algorithm completes.

Thus, $m+t \geq \lceil \log_{5/4}(n/2) \rceil + \lceil \log_2 \log_2(2n/3) - \log_2 \log_2(4/3) \rceil$ guarantees completion. In particular, choosing $m + t = \lfloor 5 \ln n \rfloor \geq \lceil \log_2 \log_2(2n/3) - \log_2 \log_2(4/3) \rceil$ successes guarantees completion. (The last inequality was verified by computer.) Chernoff's bound answers the question of how many overall iterations (not necessarily successes) suffice, with high probability, to achieve $\lfloor 5 \ln n \rfloor$ successes. For our estimate, the previous trials can be dominated by independent Bernoulli trials, with $p = 1/3$, $\epsilon = 1/2$, and $t = 6\lfloor 5 \ln n \rfloor$. The Chernoff bound then becomes

$$\Pr(S_t \leq \lfloor 5 \ln n \rfloor) \leq e^{-5(\ln n)/4} < e^{-\ln n} = 1/n.$$

Hence, after at most $t = 6\lfloor 5 \ln n \rfloor \leq 30 \ln n$ random group elements are generated, the algorithm will have constructed the Schreier tree with probability at least $1 - 1/n$. If we replace t by $\lfloor \delta c \log_2 n \rfloor$ in the previous calculation, choose $c = \lfloor 30 \ln 2 \rfloor = 20$, and choose $\delta \geq 1$, then Chernoff's bound proves Theorem 3.5. □

Remark. In the original version of this work, the same asymptotic result was given for Theorem 3.5, but with a worse constant. We thank Babai for his suggestion to use ideas from a related result in Babai and Erdős (1982) to obtain a finer estimate.

4. Random Base Change Algorithm

We will now apply the ideas described in section 3 to building a random base change algorithm. Under these circumstances, we have a random generator $Rand(G)$ which can be built from a strong generating set relative to the old ordering.

Let $\mathcal{S}$ be a Schreier vector data structure for G relative to the ordering α. Define

$$\mu(\mathcal{S}) = \prod_i |\rho(\mathcal{S}(i))|,$$

where $\rho(\mathcal{S}(i))$ is the set of nodes in the Schreier tree $\mathcal{S}(i)$.

Random-Basechange
Input: A Random generator $Rand(G)$ for G and a new ordering α of Ω.
Output: A complete Schreier vector data structure $\mathcal{S}$ for G relative to α.
Let $\mathcal{S}$ be a trivial Schreier vector data structure
While $|\mu(\mathcal{S})| < |G|$ do
  Set $g \leftarrow Rand(G)$
  Set $(g', i) \leftarrow$ Schreier-Tree-Stabilize(g)
  If $i < n$ then (Insert a new generator at level i.)
    Update-Schreier-Tree($\mathcal{S}, i, g'$)

Denote by $\rho_\alpha(\mathcal{S}(i))$ the set of points α_j such that $j \in \rho(\mathcal{S}(i))$.

Schreier-Tree-Stabilize
Input: $g \in G$ and the Schreier vector data structure $\mathcal{S}$.
Output: Find the first level $i < n$ where $\rho_\alpha(\mathcal{S}(i))^{g'} \neq \rho_\alpha(\mathcal{S}(i))$ and return (g', i).
  Here g' is the residue of g after factoring it through the first i levels.
  Otherwise return (e, n).
Set $i \leftarrow$ first-pt-moved(g)
If $i = \texttt{NIL}$ then return(e, n)
Elseif $\rho_\alpha(\mathcal{S}(i))^g = \rho_\alpha(\mathcal{S}(i))$
  Set $g' \leftarrow gh^{-1}$ where h is a coset representative of $\mathcal{S}(i)$
   such that $\alpha_i^h = \alpha_i^g$
    ($g' \in G_{\alpha_1,\ldots,\alpha_{i-1}}$ is computed from $\mathcal{S}(i)$
     and moves α_i to α_i^g)
  Return(Schreier-Tree-Stabilize(g'))
Else return(g, i)

Update-Schreier-Tree
Input: A Schreier vector data structure $\mathcal{S}$, an index i, and $g \in G^{(i)}$.
Let $P = \rho(\mathcal{S}(i))$
Foreach $j \in P$ do
  If $\alpha_j^g = \alpha_k$ and $k \notin \rho(\mathcal{S}(i))$ then

Add k to $\rho(\mathcal{S}(i))$ and add the edge (j,k)
to $\mathcal{S}(i)$ with edge label g

Before doing the analysis of the algorithm, we require a preliminary result which is well known (see Babai, 1979).

LEMMA 4.1. *Let H be a subgroup of an arbitrary finite group G and let Γ be a complete set of right coset representatives for H in G. Let Rand(G) be a random generator on G according to the uniform distribution and let Rand(H) be a random generator on H induced by Rand(G) according to the rule, if $g = Rand(G)$, then $Rand(H) = g\gamma^{-1}$ where $Hg = H\gamma$, $\gamma \in \Gamma$. Then Rand(H) has the uniform distribution.*

PROOF. Given an arbitrary $h \in H$, there are exactly $|\Gamma|$ elements of G which will produce h, namely the elements of the set $\{h\gamma\colon\ \gamma \in \Gamma\}$. Since each element of H has the same probability of being produced, namely $|\Gamma|/|G|$, *Rand*(H) has the uniform distribution. $\square$

THEOREM 4.2. *There exists a constant $c > 0$ (one can choose $c = 20$) such that for $\delta \geq 1$, the probability is at least $1 - 1/|G|^\delta$ that Random-Basechange will complete with a Schreier vector data structure $\mathcal{S}$ with respect to the ordering α such that*

(i) at most $\delta c \log_2 |G|$ random group elements are required.
(ii) $\sum_i depth(\mathcal{S}(i)) \leq \delta c \log_2 |G|$.

PROOF. Let $\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_b}$ be the base points of G listed in order of increasing indices and let $n_j = |\alpha_{i_j}^{(G^{(i_j)})}|$. The value of n_j is not known at the beginning of Random-Basechange. We define a Bernoulli trial for each random element g generated by *Rand*(G) and processed by the algorithm. Let $(g', i_j) =$ Schreier-Tree-Stabilize(g) and let $P = \rho_\alpha(\mathcal{S}(i_j))$ prior to the procedure call. The trial is considered a *success* if

(i) $\mathcal{S}(i_j)$ is the first incomplete Schreier tree, and
(ii) $|P^g - P| > (|P||P'| - \min(|P|^2, |P'|^2)/2)/n_j$.

Otherwise the trial is considered a failure. Since n_j is not known in advance, we cannot decide during Random-Basechange whether a particular trial is successful or not. As in the proof of Theorem 3.5, the algorithm will terminate after at most

$$\sum_j 5 \ln n_j = 5 \ln |G|$$

successes.

We claim that while $\mu(\mathcal{S}) < |G|$, the probability of a successful trial is at least 1/3. To prove this, let g be a randomly generated element given by *Rand*(G) and let Schreier-Tree-Stabilize be called with input g. If $\mathcal{S}(i_j)$ is the first Schreier tree which is incomplete, then eventually we compute g' so that

$$g = g' g_{i_{j-1}} \cdots g_{i_1}$$

where g_{i_k} is a coset representative for $G^{(i_{k+1})}$ in $G^{(i_k)}$ computed from $\mathcal{S}(i_k)$, $1 \leq k \leq j-1$, and where $g' \in G^{(i_j)}$. Since each $\mathcal{S}(i_k)$ is complete for $1 \leq k \leq j-1$, it follows from Lemma 4.1, that g' has equal likelihood of being any element of $G^{(i_j)}$. But then, by Corollary 3.4, the probability is at least $1/3$, that Schreier-Tree-Stabilize will return (g', i) and that the conditions for a successful trial will be satisfied.

Note that if the trial is unsuccessful, then either Schreier-Tree-Stabilize will return (g', i) and not increase $\mathcal{S}(i_j)$ by a suitable amount or else the procedure will continue with the base point $\alpha_{i_{j+1}}$, which may or may not lead to useful work.

We now use the Chernoff bounds for $p = 1/3$, $\epsilon = 1/2$, and $t = 6\lfloor 5 \ln |G| \rfloor$ trials as before, to show that for some constant c (one can choose $c = 20$) and arbitrary $\delta \geq 1$, the probability is at least $1 - 1/|G|^\delta$ that after $\delta c \log_2 |G|$ random elements are processed, there will be at least $\lfloor 5 \ln |G| \rfloor$ successful trials. The proof of (i) and (ii) now follows directly. □

Proof. (*Proof of Theorem A*) If we set $\delta = 1$ in the statement of Theorem 4.2, then conditions (i) and (ii) together with $c = 20$ certify that $\mathcal{S}$ is a short Schreier vector data structure. □

Proof. (*Proof of Theorem B*) We will assume in the statement of Theorem 4.2 that $\delta = 1$ and that (i) and (ii) are true. Consider the cost for processing each random element. The cost of generating each random element is $O(n \log |G|)$ by assumption. The cost of applying Schreier-Tree-Stabilize is $O(n \log |G|)$, by Theorem 4.2(ii). Finally, there is the cost of invoking Update-Schreier-Tree if necessary, which is $O(n)$. Thus the cost of processing each random element is $O(n \log |G|)$. Since $O(\log |G|)$ elements are processed by Theorem 4.2(i), this gives the time bound of $O(n \log^2 |G|)$ as required. The analysis when the cost of generating random elements is $O(f(n, |G|))$ is identical. □

5. An Efficient Implementation

In this section we present a more efficient Random-Basechange algorithm which satisfies the conclusion of Theorem 4.2 and is very fast in practice for small b. This version of the algorithm is similar to the one presented in section 4. The primary exception is the method used to update the Schreier tree $\mathcal{S}(i)$ whenever a new element g is used to enlarge the tree. The basic idea is to rebuild the Schreier tree $\mathcal{S}(i)$ using all available edge labels of $\mathcal{S}(j)$, $j \geq i$, allowing the depth of the tree to increase by 1. The net effect is that fewer generators are required in order to complete the algorithm and the resulting Schreier trees are shallower. This is all accomplished within the same theoretical time bounds.

Update-Schreier-Tree-Modified
Input: A Schreier vector data structure $\mathcal{S}$, an index i, and $g \in G^{(i)}$.
[Update $\mathcal{S}(i)$, increase it's depth by 1]
Set $d \leftarrow depth(\mathcal{S}(i))$
Set $T \leftarrow g \cup$ the set of edge labels of all $\mathcal{S}(j)$, $j \geq i$.
Set $\mathcal{S}(i) \leftarrow$ Build-Schreier-Tree$(i, T, d+1)$

Update-Schreier-Tree-Modified uses the procedure Build-Schreier-Tree. The input to Build-Schreier-Tree is a level i, a set of generators $T \subseteq G^{(i)}$ used to build the tree, and a depth to which the tree is to be built. Build-Schreier-Tree builds $\mathcal{S}(i)$ using a breadth-first search.

Build-Schreier-Tree
Input: An index i, $1 \leq i < n$, a list of generators $T \subseteq G^{(i)}$
and an integer d which limits the depth of the Schreier tree being built.
Output: A Schreier tree for the base point i with depth $\leq d$.
[Build the Schreier tree breadth first]
Initialize $\mathcal{T}$ to be a Schreier tree with root i.
Set *open_set* $\leftarrow \{i\}$
While *open_set* $\neq \emptyset$ do set $j \leftarrow$ first point in *open_set*
Remove j from *open_set*
If the depth of j is $< d$ then
For $g \in T$ do
If $\alpha_k = \alpha_j^g$ and $k \notin \rho(\mathcal{T})$ then
[k is now a new node in the Schreier tree with label g and parent j].
Set $\rho(\mathcal{T}) \leftarrow \rho(\mathcal{T}) \cup \{k\}$
Append k to the end of *open_set*
Else Return ($\mathcal{T}$)
Return ($\mathcal{T}$)

Let this version of the random base change algorithm be called Random-Basechange-Efficient. When Update-Schreier-Tree-modified updates the Schreier tree $\mathcal{S}(i)$ it rebuilds the entire tree using all the edge labels of $\mathcal{S}(j)$, $j \geq i$, by a breadth-first search, only up to a depth of 1 more than the previous depth of $\mathcal{S}(i)$. As Build-Schreier-Tree uses all the previous edge labels of $\mathcal{S}(i)$ along with the new generator, the updated tree has all the points that were previously in the tree, as well as 1 or more new points due to the new generator. Since Build-Schreier-Tree does a breadth-first search, the additional edge labels of $\mathcal{S}(j)$, $j > i$, that it uses may help to decrease the depth of the updated tree and increase the number of points in the tree. This results in obtaining shorter trees and completion of the algorithm using fewer generators.

THEOREM 5.1. *The procedure Random-Basechange-Efficient satisfies the same properties as Random-Basechange in Theorem 4.2. Further, if $\delta = 1$ in the hypotheses of Theorem 4.2, then with probability at least $1 - 1/|G|^\delta$ the running time for Random-Basechange-Efficient, aside from the calls to Rand(G), is $O(n \log^2 |G|)$.*

PROOF. It was noted in the previous paragraph, that each call to Update-Schreier-Tree-Modified can only increase the number of points in a Schreier tree as compared to the procedure Update-Schreier-Tree, without increasing the depth of the tree. Therefore, the proof of Theorem 5.1, will follow, aside from the timing statment, exactly as the proof of Theorem 4.2. The timing statement follows as in the proof of Theorem B, with the exception of the time required for Update-Schreier-Tree-Modified. Therefore, it suffices to show that the overall time spent on calls to Update-Schreier-Tree-Modified is $O(n \log^2 |G|)$. To see this, first note that if the event of Theorem 4.2 occurs, then at most $O(\log |G|)$ elements are processed during the course of the algorithm. This leads to at most $O(\log|G|)$ calls to Update-Schreier-Tree-Modified. Furthermore, the cost of each call to this procedure is $O(n \log|G|)$, since at most $c \log|G|$ generators can be used to rebuild any Schreier tree. The claim now follows directly. □

A slight optimization to this version of the random base change algorithm can be effected by using two constants, *init-depth* (initial depth) and *inc-depth* (incremental depth). The first call to Build-Schreier-Tree for the i^{th} Schreier tree from Update-Schreier-Tree-Modified is of the form Build-Schreier-Tree($i, T,$ *init-depth*). Thereafter, each call is of the form Build-Schreier-Tree($i, T, d+$ *incr-depth*), where d is the current depth of the Schreier tree. It is clear that the use of these two constants has the potential effect of changing only the constants of proportionality for the conclusion of Theorem 5.1. A prudent choice of values for these parameters can improve both the running time and the number of generators required to build the Schreier vector data structure.

Table 1 reports the running times for the random base change algorithm applied to several test groups using three choices of settings for *init-depth* and *inc-depth*. For each group, the first row corresponds to *init-depth* and *inc-depth* set to 0 and 1, respectively. The second and third rows correspond to settings of $(0,3)$ and $(2,3)$ for those parameters. For each group G a strong generating set and corresponding Schreier vector data structure is first built. The base change algorithm is run once, prior to any timing, to obtain a new Schreier vector data structure whose elements are independent of the method used to obtain the first one. Thus, the input and output Schreier vector data structures of the base change algorithm will have similar characteristics for the purpose of the timing. The times found will break down into two roughly equal parts: the time to generate new random elements, and the time to build the Schreier vector data structures by sifting them. The first half of the timing may vary in practice, depending on the particular data structure used to generate random group elements.

The generators in GAP are drawn from various sources in the literature. Both CAYLEY (Cannon, 1984) and GAP (Schönert *et al.*, 1993) provide excellent examples of generators for interesting groups. In addition, some generators for the higher degree groups were obtained from the work of J. Leon. The new ordering α of the underlying point set is randomly chosen. The tests were run on a SPARCstation-2 workstation using AKCL COMMON LISP (version 1.615), with permutation multiplication programmed in C. The times and number of random elements have been averaged over 10 runs. The current numbers are not directly comparable to our previously reported results, since they are the result of a more efficient implementation. Nevertheless, the same trends as before are apparent. The scaled theoretical time estimate of $n \log^2|G|/10^6$ has been included in the table to show the excellent agreement with the experimental times.

Acknowledgement. The authors are grateful to László Babai for many insightful comments on an earlier version of this paper.

Group G	degree n	base size	$\log_2 \lvert G \rvert$	$n(\log_2 \lvert G \rvert)^2$ $/10^6$	max. depth Schreier tree	gens stored	time (s)
Rubik's Cube	48	18	65.2	0.20	4.5	37.6	0.62
					4.9	32.1	0.50
					4.9	30.8	0.49
Rubik's Cube	96	52	183.5	3.23	4.4	104.7	6.62
$(4 \times 4 \times 4)$					4.8	87.0	5.04
					5.0	80.6	4.67
Cappel 1	100	8	22.9	0.05	5.9	14.2	0.18
					7.2	9.4	0.11
					6.9	9.4	0.11
Cappel 2	100	9	25.2	0.06	5.6	16.0	0.21
					6.6	10.3	0.13
					6.2	10.8	0.13
Cappel 3	72	18	44.6	0.14	5.0	27.0	0.44
					5.0	24.3	0.41
					5.0	24.6	0.39
MCL	275	5	29.7	0.24	5.6	13.5	0.34
					6.1	10.5	0.27
					6.3	9.7	0.24
$G_2(3)$	351	4	22.0	0.17	5.8	10.3	0.27
					6.4	8.5	0.24
					6.4	7.6	0.23
$G_2(4)$	416	7	27.9	0.32	5.0	14.8	0.54
					5.6	11.7	0.41
					5.7	11.3	0.43
$PSP_6(3)$	364	8	32.1	0.38	5.2	16.7	0.60
					6.7	14.8	0.55
					6.1	13.8	0.48
$U_6(2)$	672	6	33.1	0.74	5.5	18.0	1.19
					5.9	15.3	1.04
					6.0	14.9	0.99
Suzuki	1782	7	38.7	2.67	5.9	16.8	2.81
					6.1	13.4	2.95
					6.5	12.2	1.96
PSU(7,2)	2709	7	47.7	6.16	5.7	21.1	6.26
					6.1	18.2	6.03
					6.2	16.0	4.57
Fi(22)	3510	7	45.9	7.39	6.0	19.7	9.60
					6.2	15.7	7.37
					6.5	15.2	6.77
Omega+(84)	5525	8	55.9	17.26	6.0	22.9	18.27
					6.4	19.2	17.61
					6.4	19.0	15.39
FI(23)	31671	9	61.8	121.06	6.3	23.8	222.12
					7.0	18.6	174.35
					7.0	17.8	145.59
$PSp_4(47)$	106080	6	54.5	315.61	8.3	22.8	607.49
					11.6	16.3	450.43
					11.6	15.7	415.40

Table 1. Measured times for the random base change algorithm for various groups. (*init-depth, max-depth*) set to (0,1), (0,3) and (2,3) for each group.

References

Babai, L. (1979) Monte-Carlo Algorithms in Graph Isomorphism Testing. Université de Montréal Tech. Report D.M.S. 79-10, Dep. Math. et Stat.

Babai, L., Cooperman, G., Finkelstein, L., Luks, E.M., and Seress, Á. (1991) Fast Monte Carlo Algorithms for Permutation Groups. *Proc. 23rd ACM STOC*, 90–100.

Babai, L., Erdős, P. (1982) Representation of Group Elements as Short Products. *Annals of Discrete Mathematics* **12**, 27–30.

Babai, L., E. Luks, and Seress, Á. (1988) Fast Management of Permutation Groups, *Proc. 29th IEEE FOCS*, 272–282.

Brown, C.A., Finkelstein, L., and Purdom, P.W. (1989) A New Base Change Algorithm for Permutation Groups. *SIAM J. Computing* **18**, 1037–1047.

Brown, C.A., Finkelstein, L., Purdom, P.W. (1988) Backtrack Searching in the Presence of Symmetry. *Proc. of the 6th International Conference on Algebraic Algorithms and Error Correcting Codes* (AAECC-6, Rome), Springer Verlag Lecture Notes in Computer Science **357**, 99–110.

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

Butler, G., Lam, C. (1985) Isomorphism Testing of Combinatorial Objects. *J. of Symbolic Computation* **1**, 363–381.

Cannon, J.J. (1984) An Introduction to the Group Theory Language, Cayley. in *Computational Group Theory*, edited by M.D. Atkinson, New York: Academic Press, 145–184.

Chernoff, H. (1952) A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. *Annals of Math. Statistics* **23**, 493–507.

Cooperman, G., Finkelstein, L. (1992) New Methods for Using Cayley Graphs in Interconnection Networks. *Discrete Applied Mathematics* **37/38** (special issue on Interconnection Networks), 95–118.

Cooperman, G., Finkelstein, L. (1991) A Strong Generating Test and Short Presentations for Permutation Groups. *J. Symbolic Computation* **12**, 475–497.

Cooperman, G., Finkelstein, L. (1992) A Fast Cyclic Base Change for Permutation Groups. *Proc. of International Symposium on Symbolic and Algebraic Computation* (ISSAC-92), New York: ACM Press, 224–232.

Cooperman, G., Finkelstein, L., Luks, E. (1989) Reduction of Group Constructions to Point Stabilizers. *Proceedings of the International Symposium on Symbolic and Algebraic Computation* (ISSAC 89), New York: ACM Press, 351–356.

Cooperman, G., Finkelstein, L., Purdom, P.W. (1989) Fast Group Membership using a Strong Generating Test for Permutation Groups. *Computers and Mathematics*, (E. Kaltofen and S.M. Watt, eds.) New York: Springer-Verlag, 27–36.

Cooperman, G., Finkelstein, L., Sarawagi, N. (1990) A Random Base Change Algorithm for Permutation Groups. *Proc. of 1990 International Symposium on Symbolic and Algebraic Computation*, (ISSAC 90, Tokyo), New York: ACM Press and Addison-Wesley, 161–168.

Jerrum, M. (1986) A Compact Representation for Permutation Groups. *J. Algorithms* **7**, 60–78.

Knuth, D.E. (1991) Notes on Efficient Representation of Perm Groups. *Combinatorica* **11**, 57–68 (preliminary version circulated since 1981).

Lam, C., Thiel, L. (1989) Backtrack Search with Isomorph Rejection and Consistency Check. *J. Symbolic Computation* **7**, 473–485.

Leon, J. (1984) Computing Automorphism Groups of Combinatorial Objects. in *Computational Group Theory*, edited by M. D. Atkinson, Academic Press, 321–337.

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

Sims, C.C. (1971) Computation with Permutation Groups. in *Proc. Second Symposium on Symbolic and Algebraic Manipulation*, edited by S.R. Petrick, New York: ACM Press, 23–28.