# COMPUTABLE SUBGROUP CHAINS AND SHADOWING

GENE COOPERMAN AND SCOTT H. MURRAY

Abstract. We present a new structural framework for computational group theory, based on chains of subgroups. This extends existing methods, such as Schreier-Sims techniques for permutation groups. This framework is now a part of the GAP 4 computational algebra system. It will be useful for implementing the matrix group recognition project.

Contents



## 1. Introduction

In this paper, we introduce a new framework for finite group computation, called *computable subgroup chains*. This framework requires an object-oriented language and so we have implemented it in GAP 4. Divide and conquer is an archetypal method for tackling computational problems—in computational group theory, we usually use subgroup chains to acheive this. In many older algorithms, such as Schreier-Sims, all the groups that appear are represented as subgroups of single permutation or matrix group. This is not true of newer algorithms, such as matrix group recognition [18], structure forest algorithms [4], Luks' algorithm for soluble matrix groups [20], and Kantor's Sylow subgroup algorithms [13, 14]. Our framework supports standard algorithmic techniques (such as sifting and shadowing) in a consistent manner regardless of the representations used for subgroups and quotients.

Subgroup chains are used in the analysis of many algorithms for groups, but their existence is often obscured in the implementation because it was difficult to express the concepts involved explicitly in the standard computer algebra systems (Magma and GAP). A good example is provided by Schreier-Sims techniques, which

are the basis of most permutation group algorithms [24]. These techniques involve a chain of stabiliser subgroups but this chain is stored implicitly as a base and strong generating set rather than explicitly as a subgroup chain. Our new computable subgroup chains provide a highly modular framework for group computation which is as close as possible to how theorists think about groups. This facilitates the development and implementation of new algorithms and the interlacing of existing algorithms. For example, one of the oldest techniques in computational group theory is sifting (or stripping) an element through a stabiliser chain [25]. This idea has also been used in computations with other kinds of subgroup chain [18], but has not been implemented in general. In fact, sifting is not supported for chains with a mixture of different kinds of subgroup, even if it is supported for each particular kind of subgroup involved. Our framework is especially useful for dealing with mixed chains—previously algorithms involving more than one kind of subgroup, such as Luks' algorithm, tended to be too technically complicated for practical implementation.

We frequently use the concept of "computability." Our primary concern is *practical computability*—does the algorithm work in a reasonable time on the kinds of examples we are interested in? However there are more precise models of computability, of which *polynomial time* is the most common.

Algorithms for computing with large groups generally depend on sampling random elements of the group. These algorithms can be of two types: a *Monte Carlo* algorithm is guaranteed to be correct with a certain (arbitrarily high) probability, while a *Las Vegas* algorithm either returns a correct answer with a certain probability or it fails. We are primarily interested in Las Vegas algorithms, because the users of practical implementations cannot be expected to deal with the possibility that the output is simply wrong. Many Las Vegas algorithms are a combination of a Monte Carlo algorithm and a deterministic *verification* algorithm to check the correctness of the result. The product replacement method [8] or a variant of it [17] is used in practice to generate random elements of a group; it was shown to be polynomial time by Pak [22].

In Section 2, we describe shadowing and its relationship to straight-line-program techniques. This allows us to give a general description of a computable subgroup chain in Section 3, which captures the precise properties that a chain needs in order to be useful for the kinds of computation we have in mind. We describe the two main kinds of subgroups that appear in our chains in Sections 3.1 and 3.2, and verification techiniques in Section 3.3. The remainder of the paper describes applications of our framework: Section 4.1 discusses new ideas for permutation groups; Section 4.2 is an application to the matrix group recognition project; Section 4.3 discusses the use of nonstandard actions. Our implementation of the Luks soluble matrix group algorithm will be described in a forthcoming paper by Cooperman, Murray and O'Brien.

## 2. Shadowing and straight line programs

Suppose we have a group $G$ with generating set $X$, and a computable homomorphism $\varphi$ from $G$ onto another group $\overline{G}$. We assume that $\overline{G}$ is more amenable to computation than $G$. In this section we describe methods for deriving information about $G$ by doing computations in $\overline{G}$. In particular, we need to be able to find

generators for the kernel $K = \ker\varphi$, so that we can continue working down our subgroup chain. Write $\overline{g} = \varphi(g)$ for $g \in G$ and $\overline{X} = \varphi(X)$ for $X \subseteq G$.

A typical example is when $G \leq \mathrm{GL}(V)$ is imprimitive; that is, $V$ has a $G$-invariant direct sum decomposition $U_1 \oplus \cdots \oplus U_k$. The action of $G$ on this decomposition induces a map $\varphi : G \to S_k$. The image of an element of $G$ can be computed efficiently and the permutation group $\overline{G}$ is much more amenable to computation that the original matrix group.

We want to pull information back to our group $G$ by doing computation in $\overline{G}$. Since $\overline{G}$ is, by definition, the set of all words in $\overline{X}$, any computation in $\overline{G}$ will inevitably involve computing such words. If a computation in $\overline{G}$ results in a word $w(\overline{X})$, we can pull information back into $G$ by evaluating the word $w(X)$; for example, if the computation produces a relation for $\overline{G}$ (i.e., $w(\overline{X}) = 1$), then the pullback $w(X)$ will be an element of the kernel. This idea has been used in several published algorithms [4, 13, 14, 20, 18], but it is rarely used in practice because of the difficulty of going through large amounts of code and adding to each group operation corresponding code for a word equivalent.

*Shadowing* is a new method for solving this implementation problem. Like most object oriented languages, GAP 4 allows different representations to be used for a given kind of object; for example, matrices can have dense and sparse representations. Given $g$ in $G$, the image $\overline{g}$ corresponds to the coset $Kg$. A *hom-coset* is the pair $(g, \overline{g}) \in G \times_\varphi \overline{G}$ considered as a representation of $\overline{g}$. All group operations for hom-coset are defined coordinatewise, but group actions are defined for $\overline{g}$ only (i.e., $x^{(g,\overline{g})} := x^{\overline{g}}$ for $x$ in some $\overline{G}$-set). This means that all existing programs that work in the group $\overline{G}$ will also work in the group of hom-cosets *with no change to existing code*, but the hom-cosets will automatically keep track of the corresponding elements in the preimage $G$.

Of course we don't want every operation in $\overline{G}$ to be shadowed in $G$. For example, computing a strong generating set with Schreier-Sims methods is much more efficient if a base and the order of the group is known. So we can start by computing the base and order of $\overline{G}$ without shadowing, then recompute the strong generating set with shadowing in many fewer operations.

However careful we are, shadowing $\overline{G}$ by $G$ is likely to involve some unecessary operations in $G$. The alternative is to shadow elements of $\overline{G}$ by words in $X$, and only evaluate the words as elements of $G$ when absolutely necessary. This requires the use of straight line programs (SLPs) to represent words [4], since the size of an SLP grows linearly in the number of operations while the traditional representation for words can grow exponentially. Hom-cosets allow such SLPs to be computed with no change to existing code—the overhead of doing one SLP operation for every operation in $\overline{G}$ is trivial.

## 3. Computable subgroup chains

In this section, we describe the core of our framework for group computations. Before discussing chains, we consider the properties needed for each subgroup in a chain. Let $H$ be a subgroup of $G$ and let $\Delta$ be a set which indexes the cosets of $H$ in $G$. Suppose we can compute two functions:

- the *index function* $G \to \Delta$ which takes any element $g$ to the index in $G$ of the coset $Hg$; and

- the *representative function* $\Delta \to G, \delta \mapsto g_\delta$ which takes any index to a unique representative of the corresponding coset.

We now say that the subgroup $H$ has a *computable transversal* in $G$. Note that the set $\Delta$ is included largely for convenience, since there are natural indexing sets for the transversals in the two most common cases. We could consider the cosets to be indexed by their representatives, so that we have a single function $G \to G$ which takes every element to the unique representative of its coset.

We now define a *computable subgroup chain* as a chain of subgroups

$$G = G_1 \geq G_2 \geq \cdots \geq G_s \geq G_{s+1} = 1,$$

where each $G_{i+1}$ has a computable transversal in $G_i$ with index set $\Delta_i$.

Suppose $g$ is in $G$. Then we can compute the index $\delta_1$ for the coset $G_2 g$, so $g = h g_{\delta_1}$ where $h$ is a uniquely determined element of $G_2$. Iterating this process we get

$$g = g_{\delta_k} \cdots g_{\delta_2} g_{\delta_1}$$

where the sequence $(\delta_1, \ldots, \delta_k)$ is uniquely determined by $g$. This technique for decomposing $g$ is called *sifting* (or stripping). It is the fundamental operation within our framework and is already used extensively for Schreier-Sims techniques.

Given a computable subgroup chain, we can do several basic computations:

- The order of $G$ is just the product of the sizes of the index sets $\Delta_i$.
- We can generate a uniformly distributed random element of $G$ by picking a random element $\delta_i$ from each index set $\Delta_i$ and then forming the product $g_{\delta_k} \cdots g_{\delta_2} g_{\delta_1}$.
- We can iterate the elements of $G$ by iterating the sequences $(\delta_1, \ldots, \delta_k)$. This allows us to do a backtrack search in the group [6].

The *membership problem* is one of the most important in computational group theory: given an object of the correct type (e.g., a permutation of the same degree as our permutation group) we wish to determine if it is an element of $G$. This can also be done by sifting: if the element is in $G$, sifting provides a proof of that fact; if the element is not in $G$ then sifting fails in a predictable way. Failure can only occur in two ways: either the index of a coset cannot be computed or we end up with a nontrivial remainder in $G_{s+1} = 1$.

The sifting algorithm can also fail while constructing a chain with a Monte Carlo algorithm. In this case we get some remainder, which is called the *sifted* version of our original element. This is frequently useful because it allows us to generate elements further down in the chain.

In the following two sections, we discuss the two main kinds of computable transversal that appear in computational chains: transversals by Schreier tree and transversals by homomorphism. In both cases, the main difficulty is computing generators for the subgroup, which is necessary in order to iterate the algorithm. In practice these generators are computed by randomised Monte Carlo techniques together with a verification algorithm to test correctness. See Section 3.3 for a discussion of verification algorithms.

3.1. **Transversals by Schreier tree.** Schreier-Sims techniques for permutation groups are among the most successful in computational group theory. They are based on the computation of a chain of stabiliser subgroups. This project grew

out of an attempt to implement a version of the Schreier-Sims algorithm in GAP 4 which was able to handle matrices, permutations, and mixed representations.

Consider a group $G = \langle X \rangle$ acting on the set $\Omega$. Let $\alpha$ be a point in $\Omega$ and let $H$ be the stabiliser $G_\alpha$. The most common example is a permutation group acting on the set $\Omega = \{1, \ldots, n\}$. Other examples are a matrix group acting on vectors or subspaces, and a group acting on itself by conjugation. Note that we do not assume that the action is faithful. Most existing implementations of the Schreier-Sims algorithm only allow a limited range of actions—we have allowed arbitrary actions in the belief that many different actions are likely to be useful (see Section 4.3).

We can compute the orbit of $\alpha$ under the action of $G$ by the orbit algorithm:

```
let Δ = [α].
for δ in Δ do
   for x in X do
      if δ^x not in Δ then append it to Δ.
   end for
end for
```

It is easily seen that the set $\{g \in G \mid \alpha^g = \delta\}$ is a coset of $H$ for every $\delta$ in $\Delta$, and all cosets are obtained this way. Hence the orbit $\Delta$ naturally indexes the cosets of $H$ in $G$ with the index function given by $g \mapsto \alpha^g$.

Any function $u : \Delta \to G$ with the property that $\alpha^{u(\delta)} = \delta$ is a representative function. In order to compute such a function, we store a *Schreier tree* in addition to the orbit. This tree has a labelled edge $\delta \xrightarrow{x} \delta^x$ for every $\delta^x$ appended to $\Delta$ by the orbit algorithm. Suppose $\gamma \in \Delta$, then there is a unique path from $\alpha$ to $\gamma$ in our Schreier tree:

$$\alpha = \alpha_1 \xrightarrow{x_1} \alpha_2 \xrightarrow{x_2} \cdots \xrightarrow{x_l} \alpha_{l+1} = \gamma.$$

The representative function is given by $u(\gamma) = x_1 x_2 \cdots x_l$.

In order to iterate this process to construct a chain, we need to find generators for the stabiliser subgroup. This can be done using Schreier's lemma [12], but it is more efficient to take a random element of $G$, sift it through the existing chain of stabilisers, and use this sifted element as a generator. This is called the random Schreier-Sims algorithm and it is Monte Carlo. We can make it Las Vegas by adding a verification algorithm (see Section 3.3).

3.2. **Tranversals by homomorphism.** The ideas of Section 2 can be used to create computable transversals for the kernel of a homomorphism. Recall that $G = \langle X \rangle$, $\varphi : G \to \overline{G}$ is onto, and $\overline{g} := \varphi(g)$. Then the subgroup $H = \ker(\varphi)$ has a computable transversal: the elements of $\overline{G}$ index the cosets of $H$ and the indexing function is just $\varphi$. Let $X$ be the generators of $G$ with images $\overline{X}$ in $\overline{G}$. We assume for any element in $\overline{G}$ we can find a word in $\overline{X}$—this assumption is justified by the assumption that $\overline{G}$ more amenable to computation than $G$. Now, given an element $g$ in $G$, we compute $\varphi(g)$ and find a word $w(\overline{X})$ for it. Then $w(X)$ is the representative of the coset containing $g$.

We finish this section by discussing two important special cases of a transversal by homomorphism:

**Semidirect products:** If $G = H \rtimes K$ then we have an epimorphism $\Phi : G \to K$ with kernel $H$. In this case $\varphi(g)$ is actually the representative of the coset containing $g$, so the techniques above for computing representatives are not

needed. Direct products are an important special case: one application is the decomposition of abelian matrix groups in Luks' algorithm.

**Simple group recognition:** Suppose $G$ is (almost) simple. In this situation we take $H = 1$ and use a recognition algorithm to determine which almost simple group it is. Then we need a constructive recognition algorithm, which finds some standard set of generators of the group. This gives us an isomorphism from $G$ to a some standard computer representation for the group. The actual mechanisms of constructive recognition are an area of active research (see, for example, [15]).

3.3. **Using presentations to verify the chain.** In this section, we give a verification algorithm for subgroup chains created by Monte Carlo methods. The general setup for each step in our chain is as follows: $H \leq G$ with $G = \langle X \rangle$, $Y$ is a subset of $H$, every element of $Y$ can be expressed as a word (SLP) in $X$, and $\Delta$ is a (possibly incomplete) indexing set for the cosets of $H$ in $G$. We wish to verify that $H = \langle Y \rangle$.

We assume that we have a cheap test for membership of $H$—this is clearly true when $H$ is a stabiliser or the kernel of a homomorphism.

Suppose we have an incomplete set $R$ of relators for $G$ on the given generating set $X$. Initially $R$ can be taken to be the empty set. Then $\tilde{G} = \langle X|R \rangle$ is a (possibly infinite) group, and $G$ is isomorphic to a quotient of $\tilde{G}$. Let $\tilde{H}$ be the subgroup of $\tilde{G}$ generated by the words for the elements of $Y$. Now $|\tilde{G} : \tilde{H}|$ is a (possibly infinite) upper bound on $|G : H|$, while $|\Delta|$ is a lower bound. If we can show that these are equal, then $G \cong \tilde{G}$, $H = \langle Y \rangle \cong \tilde{H}$, and we are done. If we have a transversal by Schreier tree, then its index is bounded by the permutation degree of $G$, so it is often practical to compute $|\tilde{G} : \tilde{H}|$ by coset enumeration; this is the Todd-Coxeter-Schreier-Sims algorithm [19]. On the other hand, if we have a transversal by homomorphism, then we can compute the presentation of $G$ from presentations for $G/H$ and $\langle Y \rangle$ (which we assume are known by recursion), then it suffices to check that this presentation is correct by checking the relations; this is the algorithm of [18].

We can find a new relator for $R$ by sifting a random element through the chain, since sifting essentially writes an element as a word in the known generating set. On the other hand, sifting also gives us new generators, if it turns out that the chain is *not* complete.

Many tactical issues arrise when implementing this algorithm in practice. We discuss one such issue as an example. Recall that, in a chain consisting entirely of stabilisers, each subgroup is defined as a certain subset of the original group $G$, whereas in a chain of kernels each subgroup is only defined in relation to the previous one. In the former case, we can define several steps in the chain without worry about whether we have found all the generators of the intermediate groups. In the latter case, if we find a new generator for an intermediate group, all the subgroups below it are likely to be incorrect. So in a mixed chain, it is a good tactic to put a lot of effort into completing the generating set for a subgroup defined as a kernel before defining to the next subgroup

## 4. Applications

We now discuss several actual and potential applications of our subgroup chain framework. However, we believe that the greatest advantage of this framework is

in the way it allows you to mix and match different methods. For example you can use matrix group recognition techniques, until (for whatever reason) you happen to come across an action with a reasonably small orbit, then you can switch over to Schreier-Sims techniques.

The big question now becomes: which technique is going to be best for a particular group? There does not seem to be any easy way to answer this *a priori*. This is where parallelism becomes useful. The experience of [21] suggests that the best approach is likely to be much better than any other method. So we suggest that several different approaches be attempted simultaneously and the first one to return an answer be used.

4.1. **Structure forests and the O'Nan-Scott theorem.** Our framework allowed us to easily implement the structure forest approach for group membership pioneered by Babai, Luks and Seress [4]. They demonstrate a novel subgroup chain that involves normal subgroups induced by nontrivial orbits and block systems. In fact, we implement a Monte Carlo variation of the structure forest approach given in [2] that demonstrates $O(n^3 \log^c n + |S|n^2)$ Monte Carlo time for appropriate constant $c$.

Note that this method uses a mixture of transversals and homomorphisms. The homomorphisms involved are those given by intransitive and imprimitive actions. These are the first two classes in the O'Nan-Scott classification of permutation groups. It is natural to extend this idea to the other four classes. To do this we need some method for recognising which class we are in; this is straightforward for the first two classes, and possible for any class provided we already have a base and strong generating set [23]. We would like to have recognition algorithms for the other classes that do not require a base and strong generating set, but this is beyond the scope of this paper.

We now consider how to compute homomorphisms for each of the O'Nan-Scott classes, using the notation of Kleidman and Liebeck [16] with $G = \langle X \rangle \leq \mathrm{Sym}(\Omega)$, $n = |\Omega|$:

$\mathcal{A}_1$—*Intransitive.* In this case the group has more than one orbit; note that orbits can be computed easily. Fixed points (orbits of size one) can simply be removed, giving a smaller degree (and hence more efficient) representation. We can define our homomorphism by restricting permutations to a subset which is closed under the action of $G$. If the number $k$ of orbits is large, we should take this subset to be the union of half the orbits rather than a single orbit—this gives a chain of length at most $\log(k)$ rather than length $k - 1$.

$\mathcal{A}_2$—*Imprimitive.* Once again, blocks of imprimitivity can be computed easily. Given a block system $\Omega = \Omega_1 \cup \cdots \cup \Omega_k$, choose one point $\omega_i$ from each $\Omega_i$. Then the image of $g$ is the element $\overline{G}$ of $S_k$ such that $i^{\overline{G}} = j$ iff $\omega_i^g \in \Omega_j$.

$\mathcal{A}_3$—*Affine structures.* Here we have an identification of $\Omega$ with the affine space ${\mathbb{F}_q}^d$. We assume that this identification is computable. To find the image in $\mathrm{AGL}_d(q)$ of $g \in G$ we simply compute the action of $g$ on 0 and the standard basis in ${\mathbb{F}_q}^n$.

$\mathcal{A}_4$—*Wreath product.* In this case we have $G \leq H \wr K$ where $H \leq S_a$, $K \leq S_b$ and $n = a^b$. There is an identification of $\Omega$ with $H^b$, which we assume is computable. Let $h$ be a nontrivial element of $H$, let $h_i = (1, \ldots, h, \ldots, 1)$ with $h$ in the $i$th

position, and let $H_i = 1 \times \cdots \times H \times \cdots \times 1$ with $H$ in the $i$th position. Then the image of $g$ is the element $\overline{G}$ of $K$ such that $i^{\overline{G}} = j$ iff $h_i^g \in H_j$.

$\mathcal{A}_5$—*Nonabelian socle.* In this case we have $G \leq T^k.(\mathrm{Out}(T) \times S_k)$ where $n = |T|^{k-1}$. We can use the same method as for $\mathcal{A}_4$ (Of course, computing the identification of $\Omega$ with $T^{k-1}$ will be more difficult).

$\mathcal{S}$—*Almost simple.* Here we can use constructive recognition of (almost) simple groups, which is an area of much current research and is beyond the scope of this paper. cf 3.2

4.2. **Matrix group recognition.** The matrix group recognition algorithm has been an active area of research for the last decade [18]. Its aim is to find algorithms for computing with matrix groups over finite fields, using Aschbacher's theorem [1], which is a matrix group analogue of the O'Nan-Scott theorem. This fits very well into our framework, since all but two of the Aschbacher classes give a homomorphism with nontrivial kernel. In this section we describe how to compute these homomorphisms. One again we use the notation of [16]. Let $V = {\mathbb{F}_q}^n$ be the underlying vector space.

$\mathcal{C}_1$—*Stabilisers of subspaces.* First we can quotient out the subspace of fixed points. Now suppose the subspace $U$ is stabilised by $G$. Then we get a three step homomorphism chain. The first two steps are given by restriction to $U$ and projection to $V/U$. The kernel of these is a unipotent subgroup, which can easily be identified with a power commutator group (see our forthcoming paper with O'Brien for details).

$\mathcal{C}_2$—*Imprimitive.* Compute and then look at the image of one vector from each block as in $\mathcal{A}_2$.

$\mathcal{C}_3$—*Field extensions.* If the matrix group is absolutely reducible then we can extend field and use the techniques of $\mathcal{C}_1$.

$\mathcal{C}_4$—*Tensor decomposition.* We can do a change of basis so that our matrices are Dirac tensor products of matrices. We can then compute the homomorphism by linear algebra. Note that the image is not actually a matrix, but is in fact a projective matrix. hence we need algorithms for projective matrix groups—given how similar these are to matrix groups this should not be difficult.

$\mathcal{C}_5$—*Subfields.* Using the techniques of [11], we can do a change of basis so that all entries in our matrices are in a subfield. We then restrict to that subfield.

$\mathcal{C}_6$—*Symplectic normaliser.* Our subgroup $H$ has generators $x_1, y_1, \ldots, x_m, y_m, z$ and relations $[x_i, x_j] = [y_i, y_j] = [x_i, z] = [y_i, z] = 1$ for all $i$ and $j$; $[x_i, y_j]$ is 1 if $i \neq j$ and $z$ if $i = j$; and all generators are of order $l$ for some prime $l$ other than the characteristic of $\mathbb{F}_q$. Given an element of $H$ we need to be able to write it in the form $\prod_{i=1}^{m} x_i^{u_i} y_i^{v_i} z^w$. For convenience we write $x_{i+m} = y_i$. Given $g$ in $G$, let

$$x_i^g = \prod_{j=1}^{2m} x_j^{a_{ij}} z^{w_i}.$$

Then the image of $g$ is the matrix $(a_{ij})$ in $\mathrm{Sp}_{2m}(l)$.

$\mathcal{C}_7$—*Tensor imprimitive.* In this case $V$ is identified with a tensor power $W^{\otimes k}$. Let $w$ be a nonzero element of $W$, let $w_i = 1 \otimes \cdots \otimes w \otimes \cdots 1$ with $w$ in the $i$th position, and let $W_i = 1 \otimes \cdots \otimes W \otimes \cdots 1$ with $W$ in the $i$th position. Then the image of $g$ is the element $\overline{G}$ of $S_k$ such that $i^{\overline{G}} = j$ iff $w_i^g \in W_j$.

$\mathcal{C}_8$—*Classical; and* $\mathcal{S}$—*Almost simple.* These last two cases involve constructive recognition of simple groups, which is beyond the scope of this paper. cf 3.2

It is worth noting that $\mathcal{C}_2$ and $\mathcal{C}_7$ lead to permutation groups, while $\mathcal{A}_3$ in Section 4.2 leads to a matrix group. So our algorithms for these two classes of group will call each other.

4.3. **Actions on conjugacy classes and vector spaces.** In this section we discuss various actions that can be used for with Schreier tree transversal, other than the standard action of a permutation group. In most current implementations for finding a base and strong generating set using Schreier-Sims techniques, the chain is implicit, rather than being explicitly stored. This makes it very difficult to generalise this algorithm. For example for randomized Schreier-Sims in matrix groups you can oftenfind a good base point, and then achieve relatively small orbits. We wanted to use GAP 4's StabChain facility to do this, but it only understood permutation groups acting on points and could not be generalised tomatrix groups.

An important example in this discussion is the *conjugate action.* A group $G$ acts on a subgroup $H \leq G$ by conjugation. This can be generalized when $G$ has an embedding in the automorphism group of $H$, or when $H$ is an appropriate set, such as a conjugacy class of $G$. Note that in general, these actions may be *unfaithful.* However, our examples will always be for faithful actions.

Murray and O'Brien [21] and Butler [5] developed a methodology for treating a subgroup of $\mathrm{GL}_n(q)$ as acting on certain orbits in the union of the set of all vectors in $\mathbb{F}_q^n$ and the union of all vector subspaces of $\mathbb{F}_q^n$. That work used the concept of generalized eigenspaces to find vectors and subspaces that were heuristically found to have small orbits—especially for sporadic simple groups. The key to their success was the ability to heuristically find small orbits.

Cooperman, Finkelstein, Tselman and York [10] developed a methodology for treating a subgroup of $\mathrm{GL}_n(q)$ as acting on a conjugacy class. This was demonstrated by beginning with a matrix representation of Lyons's group and producing a permutation representation in the action on its smallest conjugacy class. The key to their success was the ability to heuristically find small conjugacy classes. While technically, this involved only a novel use of the conjugacy action, and not a full subgroup chain, it could be incorporated into other subgroup chains.

Luks, in a seminal paper [20], pointed out that the conjugate action of a matrix group on itself can be represented easily as a linear action. If $G < \mathrm{GL}_n(q)$, then there is an embedding of $G$ in $\mathrm{GL}_{n^2}(q)$ such that the image of $G$ in this embedding has a natural linear action on $\mathrm{GL}_n(q)$. To see this, note that for $g \in G$, and $m \in \mathrm{GL}_n(q)$, there is a natural embedding of $m$ in $\mathbb{F}_q^{n^2}$ as a vector of length $n^2$ whose entries are the original entries of the matrix $m$. Call this embedding $\theta$ and observe that $\theta$ is a vector space homomorphism for $\mathrm{GL}_n(q)$ viewed as a vector space. We require a homomorphism $\varphi\colon G \to \mathrm{GL}_{n^2}(q)$ such that

$$\theta(g^{-1}mg) = \theta(m)\varphi(g), \quad m \in \mathbb{F}_q^{n^2},\, g \in G$$

where the multiplication $\theta(m)\varphi(g)$ is multiplication of the vector $\theta(m)$ by the matrix $\varphi(g)$. It is clear from linear algebra and the fact that $\theta$ is linear that there is a unique solution $\widetilde{g} \in \mathrm{GL}_{n^2}(q)$ such that $\theta(g^{-1}mg) = \theta(m)\widetilde{g}$. We define $\varphi\colon G \to \mathrm{GL}_{n^2}(q)$ by $\varphi(g) = \widetilde{g}$ for $\widetilde{g}$ as above. It is clear that $\varphi$ is a homomorphism under this definition, and so we are done.

This is particularly striking in our context because it allows the conjugate action in $\mathrm{GL}_n(q)$, studied in [20], to be viewed as an action on vectors, as studied in [21]. In this unified context, one can compare the quality of the "small orbits" discovered using the two approaches. However, the software architecture described in this paper was required to allow us to easily compare these two actions in a single context.

4.4. **Small base groups.** A *small base group* is a member of a family of groups (where the family is usually clear from context) such that the number of base points is $O(\log^c n)$ for some constant $c$. Cameron [7] showed that all primitive permutation groups are small base groups, except the symmetric groups, the alternating groups, and nontrivial wreath products in the product action.

A key to fast computation with small base groups is maintaining a small depth for the Schreier trees. While the original result by Babai, Cooperman, Finkelstein and Seress [3] provides theoretical guarantees of nearly linear Monte Carlo time for small base group membership, we have found it advantageous to replace many of those theoretical guarantees with a different algorithmic version that is more suited to fast implementations. Many those theoretical guarantees came at the cost of using theoretically provable algorithms for producing sufficiently independent random elements. Instead, we use heuristic techniques to more quickly produce random elements [ref] that typically have a distribution closer to uniform than those suggested in the original paper. Having once departed from the original theoretical guarantees, we find it advantageous to replace other techniques of that paper by *cube Schreier trees* [9] in order to achieve fast heuristic times. The theoretical guarantees are not needed since we have a verification algorithm, but are useful as a guide to how most efficiently to complete the computation.

## References

[1] M. Aschbacher. On the maximal subgroups of the finite classical groups. *Invent. Math.*, 76(3):469–514, 1984.

[2] László Babai, Gene Cooperman, Larry Finkelstein, Eugene Luks, and Ákos Seress. Fast Monte Carlo algorithms for permutation groups. *J. Comput. System Sci.*, 50(2):296–308, 1995. 23rd Symposium on the Theory of Computing (New Orleans, LA, 1991).

[3] László Babai, Gene Cooperman, Larry Finkelstein, and Ákos Seress. Nearly linear time algorithms for permutation groups with a small base. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC '91)*, 1991.

[4] László Babai, Eugene M. Luks, and Ákos Seress. Fast management of permutation groups. I. *SIAM J. Comput.*, 26(5):1310–1342, 1997.

[5] Gregory Butler. The Schreier algorithm for matrix groups. In SYMSAC '76, *Proc. ACM Sympos. symbolic and algebraic computation*, pages 167–170, New York, 1976. (New York, 1976), Association for Computing Machinery.

[6] Gregory Butler. Computing in permutation and matrix groups. II. Backtrack algorithm. *Math. Comp.*, 39(160):671–680, 1982.

[7] Peter J. Cameron. Finite permutation groups and finite simple groups. *Bull. London Math. Soc.*, 13(1):1–22, 1981.

[8] Frank Celler, Charles R. Leedham-Green, Scott H. Murray, Alice C. Niemeyer, and E. A. O'Brien. Generating random elements of a finite group. *Comm. Algebra*, 23(13):4931–4948, 1995.

[9] Gene Cooperman and Larry Finkelstein. Combinatorial tools for computational group theory. In *Groups and computation (New Brunswick, NJ, 1991)*, pages 53–86. Amer. Math. Soc., Providence, RI, 1993.

[10] Gene Cooperman, Larry Finkelstein, Michael Tselman, and Bryant York. Constructing permutation representations for matrix groups. *J. Symbolic Comput.*, 24(3-4):471–488, 1997. Computational algebra and number theory (London, 1993).

[11] S. P. Glasby and R. B. Howlett. Writing representations over minimal fields. *Comm. Algebra*, 25(6):1703–1711, 1997.

[12] Marshall Hall Jr. *The theory of groups*. Chelsea Publishing Co., New York, 1976. Reprinting of the 1968 edition.

[13] William M. Kantor. Polynomial-time algorithms for finding elements of prime order and Sylow subgroups. *J. Algorithms*, 6(4):478–514, 1985.

[14] William M. Kantor. Sylow's theorem in polynomial time. *J. Comput. System Sci.*, 30(3):359–394, 1985.

[15] William M. Kantor and Ákos Seress. Black box classical groups. *Mem. Amer. Math. Soc.*, 149(708):viii+168, 2001.

[16] Peter Kleidman and Martin Liebeck. *The subgroup structure of the finite classical groups*. Cambridge University Press, Cambridge, 1990.

[17] C. R. Leedham-Green and Scott H. Murray. Variants of product replacement. In *Computational and statistical group theory (Las Vegas, NV/Hoboken, NJ, 2001)*, volume 298 of *Contemp. Math.*, pages 97–104. Amer. Math. Soc., Providence, RI, 2002.

[18] Charles R. Leedham-Green. The computational matrix group project. In *Groups and computation, III (Columbus, OH, 1999)*, pages 229–247. de Gruyter, Berlin, 2001.

[19] Jeffrey S. Leon. On an algorithm for finding a base and a strong generating set for a group given by generating permutations. *Math. Comp.*, 35(151):941–974, 1980.

[20] Eugene M. Luks. Computing in solvable matrix groups. In *Proceedings 33rd IEEE Symposium on the Foundations of Computer Science*, pages 111–120, 1992.

[21] Scott H. Murray and E. A. O'Brien. Selecting base points for the Schreier-Sims algorithm for matrix groups. *J. Symbolic Comput.*, 19(6):577–584, 1995.

[22] Igor Pak. The product replacement algorithm is polynomial. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000)*, pages 476–485, Los Alamitos, CA, 2000. IEEE Comput. Soc. Press.

[23] Ákos Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003.

[24] Charles C. Sims. Computational methods in the study of permutation groups. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 169–183. Pergamon, Oxford, 1970.

[25] Charles C. Sims. Determining the conjugacy classes of a permutation group. In *Computers in algebra and number theory (Proc. SIAM-AMS Sympos. Appl. Math., New York, 1970)*, pages 191–195. SIAM–AMS Proc., Vol. IV. Amer. Math. Soc., Providence, R.I., 1971.