

Advances in Group Theory and Applications
© 2025 AGTA - www.advgrouptheory.com/journal
20 (2025), pp. 65–120 ISSN: 2499-1287
DOI: 10.32037/agta-2025-002

Computational Group Theory: Bedrock and New Frontiers

Melissa Lee

Abstract

Computational group theory concerns the development, analysis and application of algorithms for groups. This survey paper, aimed at students and early career researchers, gives an overview of some of the important techniques and developments in computational group theory. In particular, the text discusses algorithms for orbits and stabilisers, permutation groups, finitely presented groups, and matrix groups. Each section also highlights "new frontiers": open problems and avenues for further research.

Mathematics Subject Classification (2020): 20-01, 20-08, 20B40, 20F05

Keywords: computational group theory; finite group; infinite group

1 Introduction

Computational group theory (CGT for short) concerns the development, analysis and application of algorithms for dealing with groups. The subject's roots predate the rise of mechanical computers, starting with work of Dehn [26] in the 1910s, who proposed an algorithm for the *word problem* in hyperbolic groups. In 1936, Todd and Coxeter [94] designed a coset enumeration algorithm for finitely presented groups, used for some decades in hand computations. In 1953, Hazelgrove implemented their algorithm on a computer in Cambridge, marking the first known

computer implementation of a group-theoretic algorithm. Versions of the Todd–Coxeter algorithm are still widely used in computational algebra packages today.

There are large variations in our ability to compute with a group, based on the way it is represented. Most research on the topic has been focused on algorithms for three types of groups: permutation groups, finitely presented groups, and matrix groups. Each type presents distinct advantages and challenges that have significantly influenced the pace of algorithmic development.

For example, in 1970, Sims made a major breakthrough in computing with finite permutation groups by introducing *bases* and *strong generating sets*, which remain the most useful tools for designing algorithms for groups of this type. In particular, the availability of these data for a permutation group allows for efficient computation of, amongst other things, membership and equality testing, homomorphisms construction, and normal closures.

In contrast, algorithms specially designed for matrix groups did not emerge until the 1990s. Moreover, efficient tools for dealing with large matrix groups over finite fields, based on computing a composition series of the group, only became widely accessible in the mid-2010s (see [1]).

The broad use and applicability of algorithms in CGT is due to the development of software systems designed for computational algebra. The late 1980s and early 1990s saw the rise of two seminal platforms. One of them is GAP (Groups, Algorithms, and Programming), developed by Neubüser's group at RWTH Aachen and first released in 1988 (see [40]). The second is Magma (see [11]), developed by John Cannon in Sydney and released in 1993. Both GAP and Magma continue to be updated today, and provide comprehensive environments for group manipulation.

There are still large open questions concerning computation in groups given in each of the three aforementioned represen-

tations, and computational group theory remains a vibrant and active field of research today.

This survey article is intended to be an introduction to computational group theory for students and early career researchers. It covers some of the pivotal algorithms that have shaped CGT over the past century, and mention some of the so-called "new frontiers" — avenues for further research. The account of the history and landscape of computational group theory given here is by no means complete, nor do we intend it to be. We direct the interested reader wanting to learn more to the excellent Handbook of Computational Group Theory (see [50]), as well as several books and more recent survey articles mentioned throughout the notes.

The article is organised as follows. We begin with some preliminaries in Section 2, before exploring orbits and stabiliser algorithms in Section 3, permutation groups in Section 4, finitely presented groups in Section 5, and we conclude with a discussion of matrix groups in Section 6.

2 Preliminaries and assumptions

We begin by discussing some of the basic considerations needed when computing with groups. As mentioned in the introduction, there are three primary ways to represent a group on a computer:

1. As a permutation group on a finite set Ω (preferably denoted by $\Omega = \{1, \ldots, n\}$).

2. As a matrix group over a ring, preferably where elements of the ring can be expressed exactly (e.g., $\mathbb{F}_q$ or $\mathbb{Q}$, not $\mathbb{R}$ or $\mathbb{C}$).

3. As a finitely presented group $\langle X \mid R \rangle$, where the sets of generators X and of relators R are both finite.

Due to space constraints, it is preferable to avoid enumeration of group elements. Hence, groups will usually be given by a finite generating set S, and we aim to develop algorithms that are based on operations on S. It is also often advantageous to express an element of a group as a word in the given generating set. In the case of matrix groups, such a word in the generators is commonly stored as a *straight line program*, which efficiently constructs it using only multiplication and inversion within the group. Note that it is not always possible to avoid enumeration; it is needed for example if we want to construct a regular representation of a group.

We do not include a discussion about the actual encoding of groups in computer memory here, apart to mention that it differs based on the chosen representation. However, throughout these notes we will assume that we are able to not only encode the group, but multiply and invert elements according to the group operation. In the cases of permutation and matrix groups, where there is a natural underlying set Ω on which the group acts, we will also assume that we can apply the group action on Ω. More information about group encodings can be found in [50, Section 3].

3 Orbits and stabilisers

The foremost motivation and the main application of groups is as expressions of the symmetries of objects. Orbits and stabilisers of elements within these objects are fundamental concepts in group theory, hence, so are the algorithms that compute them. These algorithms can be applied to a group G of any kind acting on a domain Ω, as long as G has a finite generating set, we are able to multiply, invert and compare elements of G, and we can apply the group action of G on Ω.

We will assume that G is given as a set of generators S and, if G is infinite, that S is inverse-closed (i.e., $s^{-1} \in S$ if $s \in S$).

We begin with a description of the basic orbit algorithm, which relies on the following simple lemma.

Lemma 1 *Let* $G = \langle g_1, g_2, \ldots, g_k \rangle \leqslant \mathrm{Sym}(\Omega)$. *Suppose* $\omega \in \Omega$ *and* $\Delta \subset \Omega$ *such that*:

(1) $\omega \in \Delta$;

(2) *for every* $\delta \in \Delta$, *there is a sequence of indices* $i_1, i_2, \ldots i_m$ *in the set* $\{1, 2, \ldots, k\}$ *such that* $\delta = \omega^{g_{i_1} g_{i_2} \cdots g_{i_m}}$;

(3) *for all* $\delta \in \Delta$ *and all generators* g_i, $\delta^{g_i} \in \Delta$.

Then $\Delta = \omega^G$.

Proof — Conditions (1) and (2) imply that $\Delta \subset \omega^G$. Moreover, condition (3) implies that Δ is a union of orbits, so we must have $\Delta = \omega^G$. □

The basic orbit algorithm, outlined in Algorithm 1, proceeds as follows. We start by initialising the orbit Δ with just one point ω. For each element δ of Δ (which is being updated as we go), we apply each group generator g_i to δ. If a given image δ^{g_i} is not in Δ, then we append it to the end of Δ. Otherwise, we do nothing. This process continues, element by element, until all elements of Δ have been processed. Once we reach the end of Δ, the conditions of Lemma 1 are satisfied, and the final orbit is returned.

For $\omega \in \Omega$ and for each $\alpha \in \omega^G$, we may also want to compute a (representative) group element $g \in G$ such that $\omega^g = \alpha$. A list of such representatives for a whole orbit ω^G is called a *transversal*.

The following lemma follows from the proof of the orbit-stabiliser theorem.

Lemma 2 *Let* G *be a group acting on a domain* Ω *and* $\omega \in \Omega$. *A transversal of* ω^G *is a list of coset representatives for* $\mathrm{Stab}_G(\omega)$.

Algorithm 1 Basic orbit algorithm

Inputs: A group G, given as a generating set $\{g_1, \ldots, g_k\}$ acting on some set/domain Ω, and $\omega \in \Omega$.
Output: The orbit $\Delta = \omega^G$.
Initialize:
    $\Delta = [\omega]$
for $\delta \in \Delta$ **do**
    for $i \in \{1, \ldots, k\}$ **do**
        $\alpha = \delta^{g_i}$;
        if $\alpha \notin \Delta$ **then**
            Append α to Δ;
        end if
    end for
end for
return Δ

In light of this lemma, we often refer to a transversal of a stabiliser — or more generally of a subgroup — within a group, in addition to a transversal of an orbit.

In order to more easily define a correspondence between the elements of an orbit Δ of G on Ω and of a transversal T of Δ, we fix an ordering of Δ, and we specify that $\omega^{T[i]} = \Delta[i]$. Moreover, when the position within the orbit/transversal is not needed, we will often abuse notation and write $T[\alpha]$ for the element of T such that $\omega^{T[\alpha]} = \alpha$.

Obtaining and storing a transversal for an orbit Δ is important, but it can be very space inefficient if the transversal is large, or if the representation of group elements itself is inefficient. For efficiency's sake, it would also be desirable to compute a transversal at the same time as the orbit, if only a little further computation is required. Both of these issues are taken care of by the introduction of *Schreier vectors*.

Definition 3 Let $\Delta = \omega^G$, considered as a list. A *Schreier vector* is a list S of size $|\Delta|$, indexed by members of Δ, such that:

(1) The entries of S are (pointers to) generators of G.

(2) $S[\omega] = 1$.

(3) If $S[\alpha] = g$ and $\beta = \alpha^{g^{-1}}$, then β appears before α in the orbit Δ.

Computation of Schreier vectors is an easy addition to the basic orbit algorithm — the revised version is given in Algorithm 2.

Algorithm 2 Basic orbit algorithm with Schreier vector calculation

Inputs: A group G, given as a generating set $\{g_1, \dots g_k\}$ acting on some set/domain Ω, and $\omega \in \Omega$.
Output: The orbit $\Delta = \omega^G$ and the Schreier vector S.
Initialize:
  $\Delta = [\omega]$;
  $S = [1]$;
for $\delta \in \Delta$ **do**
  for $i \in \{1, \dots, k\}$ **do**
    $\alpha = \delta^{g_i}$;
    if $\alpha \notin \Delta$ **then**
      Append α to Δ;
      Append g_i to S;
    end if
  end for
end for
return Δ, S

We shall now provide an instructive, if elementary, example.

Example 4 Let $G = \langle g_1, g_2 \rangle$, where $g_1 = (1\,2)$ and $g_2 = (2\,4)$. We determine the orbit of $\omega = 1$ under G using Algorithm 2, along with the Schreier vector of the corresponding transversal. We start with $\Delta = [1]$ (the orbit) and $S = [1]$ (the Schreier vector).

First, we apply g_1 to 1, yielding $1^{g_1} = 2$. Since $2 \notin \Delta$, we append 2 to Δ and g_1 to S, making $\Delta = [1, 2]$ and $S = [1, g_1]$. Next, applying g_2 to 1 gives $1^{g_2} = 1$, but since $1 \in \Delta$, no changes are made.

Applying g_1 to 2 results in $2^{g_1} = 1$, which is already in Δ.

Then, applying g_2 to 2 gives $2^{g_2} = 4$. As $4 \notin \Delta$, we append 4 to Δ and g_2 to S, resulting in $\Delta = [1, 2, 4]$ and $S = [1, g_1, g_2]$.

Finally, applying g_1 and g_2 to 4 yields no new elements, as both map to values already in Δ. Thus, since the conditions of Lemma 1 are satisfied, the orbit of $\omega = 1$ is $\Delta = [1, 2, 4]$, with $S = [1, g_1, g_2]$ and these lists are returned.

Finally, Algorithm 3 demonstrates the straightforward procedure allowing us to obtain a transversal element from a Schreier vector, making it a desirable method for storing representatives. Note that Algorithm 3 is always guaranteed to terminate by part (3) of the definition of a Schreier vector.

Algorithm 3 Schreier vector to transversal algorithm

Inputs: An element $\omega \in \Omega$, $\alpha \in \omega^G$ and a Schreier vector S for ω.
Output: A representative g such that $\omega^g = \alpha$.
Initialize:
    $\delta = \alpha$;
    $g = 1$;
while $\delta \neq \omega$ **do**
    $h = S[\delta]$;
    $g = h.g$;
    $\delta = \delta^{h^{-1}}$;
end while
return g;

For ease of exposition, we will continue to work with transversals for the remainder of the section, rather than Schreier vectors, but in practice most algorithms use the latter representation.

The following powerful result underpins several fundamental algorithms in permutation group theory, as well as providing a method of computing stabilisers of elements.

Lemma 5 (Schreier's Lemma) *Let* $G = \langle S \rangle$ *be a finitely generated group with a finite index subgroup* H. *Suppose* $R = \{r_1, r_2, \ldots, r_s\}$ *is a set of* (*right*) *coset representatives for* H *in* G, *and fix* $r_1 = 1$. *Further,*

for $g \in G$, *write* $\overline{g}$ *for the representative* $\overline{g} = r_i$ *such that* $Hg = Hr_i$. *Then,* H *is generated by*

$$U = \{\, rs(\overline{rs})^{-1} \mid r \in R, s \in S \,\}.$$

The set U *is called a set of Schreier generators for* H.

As an aside, Schreier's Lemma has many nice consequences, apart from giving us the ability to turn our orbit algorithm into an orbit stabiliser algorithm. For example, it implies that every finite index subgroup of a finitely presented group is finitely generated. We will discuss finitely presented groups more in Section 5.

Let us observe that the orbit-stabiliser algorithm can now be described as an extension of the basic orbit algorithm by an application of Schreier's Lemma. We present it in Algorithm 4.

Algorithm 4 Orbit-stabiliser algorithm

Inputs: A group G, given as a generating set $\{g_1, \dots g_k\}$ acting on some set/domain Ω, and $\omega \in \Omega$.
Output: The orbit $\Delta = \omega^G$, the transversal T, and the orbit stabilizer Stab.
Initialize:
    $\Delta = [\omega]$;
    $T = [1]$;
    Stab = [1];
for $\delta \in \Delta$ **do**
    for $i \in \{1, \dots, k\}$ **do**
        $\alpha = \delta^{g_i}$;
        if $\alpha \notin \Delta$ **then**
            Append α to Δ;
            Append $T[\delta].g_i$ to T;
        else
            Append $T[\delta].g_i.T[\alpha]^{-1}$ to Stab;
        end if
    end for
end for
return Δ, T, Stab;

After initialising the orbit Δ, transversal T, and stabiliser Stab, the algorithm largely follows the steps of Algorithm 1, iterating through each element $\delta \in \Delta$ and applying each generator g_i of G. If δ^{g_i} does not already belong to Δ, we update both the transversal T and the orbit Δ. On the other hand, if δ^{g_i} is equal to an existing element of Δ, we can construct an element of the stabiliser of ω in G by accessing the transversal.

Remark 6 The set of Schreier generators returned by Algorithm 4 is often large and redundant. We can attempt to reduce the size using a range of methods. For example, we could include a membership test of each new Schreier generator in the existing group before adding it to the set. However, this can come with its own problems, including decreased efficiency. Depending on the representation of the group, it also may be uncertain whether membership testing is possible in the first place! Another approach is to hope that a small subset of the returned generators is enough to generate the whole stabiliser, although, depending on the application, verifying this may prove to be difficult.

3.1 New frontiers

The Orb package [69] for GAP [40], written by Müller, Neunhöffer and Noeske, and incorporating earlier ideas of Lübeck and Neunhöffer [65], provides methods for computing orbits for groups with very large degrees and orbit sizes. One of the features of orbit enumeration algorithms is, of course, that each element of an orbit must be stored in memory. Therefore, the efficiency with which we can store each of the elements plays a large role in determining the size of the problems we can solve.

The general idea behind the Orb package is the construction of the orbits of a large group G, acting on a domain Ω, as a union of suborbits of a suitable so-called *helper subgroup* $U < G$. Then, rather than storing each element of the chosen G-orbit, we instead only store a selection of representatives from

each U-orbit. In order for this to be effective, we must choose our helper subgroup U carefully. In particular, in addition to being able to calculate U-orbits (unless we plan on choosing a further helper subgroup for U), we want a way of determining canonical representative(s) of each U-orbit, and a method to compute an element of U that maps a given element $\omega \in \Omega$ to the canonical representative in ω^G.

To formalise this, let Y be a finite U-set that admits a homomorphism of U-sets $\phi : \Omega \to X$. For example, we could have $\Omega = M$, where M is an FG-module for some field F, and ϕ is the restriction map from M to a submodule M' preserved by U, considered as a FU-module. For each U-orbit in X, one point is designated as U-*minimal*. A point $x \in \Omega$ is called U-minimal if $\phi(x) \in X$ is U-minimal. When enumerating Ω, we store only the U-minimal points, rather than the whole of the orbit. Computing in X rather than Ω can also allow us to use standard orbit algorithms (such as those described in this section), if X is sufficiently small. Moreover, storing canonical representatives as elements of X rather than Ω can also afford significant space savings.

Now, it is clear that the best choice of helper subgroup U has the smallest possible number of orbits on X (i.e. it has many regular orbits). In this best case, there is a memory saving factor of around $|U|$ if compared with storing every orbit element, since only one canonical representative of each of these regular orbits is needed. Iterating this method by choosing a chain of helper subgroups increases the complexity of the input needed from the user, but has substantial impact on the memory and time requirements for the orbit enumeration calculation.

This package has been used to solve some very difficult problems. For example, Hiss, Neunhöffer and Noeske [48] used Orb to compute the 2-modular character table of the sporadic simple group Fi_{23}; this required the calculation of all but one of the double coset representatives of $Fi_{23} < GL_{1494}(2)$ with respect to a maximal subgroup of index approximately 1.3×10^9. The

authors of Orb have also demonstrated a very intricate method for enumerating the smallest non-trivial orbit of the Baby Monster group $\mathbb{B} \leqslant \mathrm{GL}_{4370}(2)$. Without the use of helper subgroups, the space required to store the elements in this orbit on a 64 bit machines is estimated to be around 7.9 terabytes, while after applying Orb with three helper subgroups, this space requirement is reduced to 414MB (see [69, Section 11.4]).

Although the Orb package is very powerful, effective use often requires delicate and sometimes difficult preparations by the user. Indeed, let us mention that the authors of the package note "[w]hile these orbit-by-suborbit algorithms are much more efficient in many cases... [t]hey are definitely not intended to be push-the-button-tools but require a considerable amount of knowledge from the pilot" (see [69, Chapter 1]). For this reason, the more complex applications of Orb involving chains of helper subgroups have largely been conducted by the authors of the package, and there is great potential for new experts in orbit enumeration to emerge.

4 Permutation groups

Permutation groups are arguably the most important class of groups, since, by Cayley's theorem, any finite group can be represented as a group of permutations. It is currently possible to compute with permutation groups up to degree above 10 million, and many of the standard algorithms rely on *bases* and *strong generating sets*, first introduced by Sims in 1970 (see [91]). We will discuss the essentials of bases and strong generating sets here; for a more comprehensive account of algorithms for permutation groups, see Seress' book [89] or [50, Section 4].

Definition 7 A *base* for a permutation group $G \leqslant \mathrm{Sym}(\Omega)$ is a sequence $B = (b_1, \ldots, b_k) \subseteq \Omega$ with trivial (pointwise) stabiliser in G.

It is easy to show that each group element is determined by its action on a base.

Lemma 8 *Let* $G \leqslant \mathrm{Sym}(\Omega)$ *with base* $B \subseteq \Omega$. *Then* $B^g \neq B^h$, *for all* $g, h \in G$ *with* $g \neq h$.

Proof — If $B^g = B^h$, then $B^{gh^{-1}} = B$. Since only the identity fixes B, we must have $g = h$. □

Given $G \leqslant \mathrm{Sym}(\Omega)$ with base $B = (b_1, \ldots, b_k)$, we define the i*th basic stabiliser* $G^{[i]}$ to be $G_{b_1, b_2, \ldots b_{i-1}}$ — the pointwise stabiliser of the first $i-1$ base points. Of course, the basic stabilisers form a subgroup chain:

$$G = G^{[1]} \geqslant G^{[2]} \geqslant \ldots \geqslant G^{[k+1]} = 1.$$

If all of the inclusions in the chain are strict, we say that B is an *irredundant* base for G. We define the i*th basic orbit* $\Delta^{[i]}$ to be the orbit of the ith base point under the ith basic stabiliser, i.e., $\Delta^{[i]} = {b_i}^{G^{[i]}}$.

The following example will be used throughout the section.

Example 9 Let $G = S_n$ and $\Omega = \{1, \ldots, n\}$. Then, a base for G must include at least $n-1$ points, since there exists a transposition switching any two points outside a base for G. Conversely, we have that any subset of $n-1$ elements of Ω forms a base. Choose $B = (1, 2, \ldots, n-1)$. With this base,

$$G^{[i]} = \mathrm{Sym}(\{i, \ldots, n\})$$

for $1 \leqslant i \leqslant n$. Moreover, for each i, the ith basic orbit is

$$\Delta^{[i]} = i^{G^{[i]}} = \{i, \ldots, n\}.$$

Definition 10 Let $G \leqslant \mathrm{Sym}(\Omega)$ with base $B = (b_1, \ldots, b_k)$. A *strong generating set* (SGS) for G relative to B is a generating set for G such that $\langle S \cap G^{[i]} \rangle = G^{[i]}$, for every $1 \leqslant i \leqslant k+1$.

For each $1 \leqslant i \leqslant k+1$, we write $S^{[i]}$ to denote $S \cap G^{[i]}$, the i*th partial strong generating set* of G with respect to S.

Example 11 We continue with the example from Example 9, with $G = S_n$. With the base being $B = (1, 2, \ldots, n-1)$, we can take $S = \{(1\ 2), (2\ 3), \ldots, (n-1\ n)\}$ as a strong generating set. Indeed, for each i with $1 \leqslant i \leqslant k$,

$$\begin{aligned}\left\langle S \cap G^{[i]} \right\rangle = \langle S \cap \mathrm{Sym}(\{i, \ldots n\}) \rangle &= \langle (i\ i+1), \ldots, (n-1\ n) \rangle \\ &= \mathrm{Sym}(\{i, \ldots, n\})\end{aligned}$$

as required.

For $1 \leqslant i \leqslant k+1$, let $U^{[i]}$ be the right transversal of $G^{[i]}$ in $G^{[i-1]}$. By the orbit-stabilizer theorem, we have

$$U^{[i]} = \left\{ u_b^{(i)} \mid b \in \Delta^{[i]} \right\},$$

where $u_b^{(i)}$ maps b_i to b.

This observation will allow us to express every group element as a product of transversal elements. We first return to our example.

Example 12 We return to our example with $G = S_n$. Now, we have that $U^{[1]}$ is the right transversal of $G^{[2]} = \mathrm{Sym}(\{2, \ldots, n\})$ in $G^{[1]} = S_n$. Since the first basic orbit is $\Delta_1^{[1]} = 1^{S_n} = \{1, \ldots, n\}$, we can choose

$$U^{[1]} = \{(1\ i) \mid 1 \leqslant i \leqslant n\}.$$

Similarly,

$$U^{[i]} = \{(i\ j) \mid i \leqslant j \leqslant n\}.$$

Let us observe that, if we are given a base B and strong generating set S for G, then we can easily recover the basic stabilisers $G^{[i]}$, basic orbits $\Delta^{[i]}$, transversals $U^{[i]}$ and subsets $S^{[i]}$ using the orbit-stabiliser algorithm introduced in Section 3. Following [50, Section 4.41], we will use Δ^* to denote the list of basic

orbits $\Delta^{[i]}$, together with the relevant data structures/methods to compute the elements $u_b^{(i)} \in U^{[i]}$ (cf. Algorithm 2).

Since bases and strong generating sets are integral for many algorithms for permutation groups, we will often refer to them as a pair with the acronym BSGS.

Lemma 13 *Let* $G \leqslant \mathrm{Sym}(\Omega)$ *and adopt the same notation as above. Every element* $g \in G$ *can be written uniquely as a product*

$$g = u_k u_{k-1} \dots u_2 u_1,$$

where $u_i \in U^{[i]}$.

Proof — We will prove the existence and uniqueness of the representation of $g \in G$. Let $B = (b_1, b_2, \dots, b_k)$ be a base for G. If $k = 1$, then the proof is complete because $U^{[1]} = G$. Assume $k > 1$. Now, by definition, we may find $u_1 \in U^{[1]}$ mapping b_1 to $b_1{}^g$. Set $g_1 = g u_1^{-1}$. Now, we have that g_1 fixes b_1, so it must lie in the second basic stabiliser $G^{[2]}$. Therefore, we can find $u_2 \in U^{[2]}$ mapping b_2 to $b_2{}^{g_1}$. Set $g_2 = g_1 u_2^{-1}$. Now, g_2 fixes b_1, b_2, so either $k = 2$ and $g = u_2 u_1$, or g_2 must lie in the third basic stabiliser $G^{[3]}$. Hence, the result follows by induction on k. □

Lemma 13 has various applications. First, it allows us to read off the order of G as the product $|U^{[1]}||U^{[2]}| \dots |U^{[k+1]}|$. The method used in the proof of Lemma 13 can also be used to test membership in permutation groups. We present an implementation of this application, which we will later integrate into another algorithm, as Algorithm 5.

The algorithm starts by taking the element g from the symmetric group $\mathrm{Sym}(\Omega)$, a base B for the group $G \leqslant \mathrm{Sym}(\Omega)$, and the data structure Δ^* defined above, as three inputs.

The initialisation step sets $h = g$, where h will hold the "part" of g that has not yet been verified for membership in G.

The main part of the algorithm is a loop that iterates over each index $i \in \{1, \dots, k\}$, where k is the length of the base B. In

each iteration, the algorithm first computes $b = b_i{}^h$, the image of the base point b_i under the permutation h. It then checks whether this b is present in the basic orbit $\Delta^{[i]}$. If b is not found in $\Delta^{[i]}$, the algorithm concludes that g is not a member of G and returns the index i where the failure occurred, along with the remaining permutation h.

Observe that, if b is found in $\Delta^{[i]}$, the algorithm finds a permutation $u_i \in U^{[i]}$ such that $b_i{}^{u_i} = b$ (note that this capability is assumed in Δ^*), and then updates h by multiplying it with u_i^{-1}, which reduces h progressively towards the identity permutation, as in the proof of Lemma 13, since $hu_i^{-1} \in G^{[i+1]}$. This process repeats for each base point. If the algorithm completes the loop without encountering any failures, it returns $k+1$ and h, where h is now the identity permutation. This indicates that g is indeed a member of the group G, as h has been reduced to the identity through valid membership operations at each step.

Algorithm 5 Membership testing algorithm

Inputs: $g \in \mathrm{Sym}(\Omega)$; B, a base for G; the data structure Δ^*.
Output: $i \in [1, \ldots, k+1]$; $h \in \mathrm{Sym}(\Omega)$.
Initialize:
    $h = g$
for $i \in [1, \ldots, k]$ **do**
    $b = b_i{}^h$;
    if $b \notin \Delta^{[i]}$ **then**
        return i, h;
    end if
    Find $u_i \in U^{[i]}$ such that $b_i{}^{u_i} = b$;
    $h = hu_i^{-1}$;
end for
return $k+1$, h;

We have illustrated that BSGS are a powerful tool for creating more efficient representations of group elements, as well as for determining properties of the permutation group. We now

discuss an algorithm for finding a base and strong generating set.

4.1 The Schreier–Sims algorithm

The Schreier–Sims algorithm, developed by Sims in [91], computes a base and strong generating set for a group. The algorithm is based on the following straightforward observation.

Lemma 14 *Let* $G \leqslant \mathrm{Sym}(\Omega)$ *be a permutation group, and consider* $B = \{b_1, \ldots, b_k\} \subseteq \Omega$ *and* $S = \{s_1, \ldots, s_{k+1}\}$ *a generating set for* G*. Then* (B, S) *is a BSGS for* G *if and only if* $S^{[k+1]}$ *is trivial and the stabiliser of* b_i *in* $\langle S^{[i]} \rangle$ *is equal to* $\langle S^{[i+1]} \rangle$*, for each* $1 \leqslant i \leqslant k$*.*

Proof — By definition, (B, S) is a BSGS for G if and only if the $(k+1)$th basic stabiliser $G^{[k+1]}$ is trivial, and

$$\langle S^{[i]} \rangle = \langle S \cap G^{[i]} \rangle = G^{[i]}$$

for $1 \leqslant i \leqslant k$. Under these assumptions,

$$\langle S^{[i]} \rangle_{b_i} = G^{[i]}{}_{b_i} = G^{[i+1]} = \langle S^{[i+1]} \rangle$$

as required.

Conversely, suppose that $S^{[k+1]} = 1$ and $\langle S^{[i]} \rangle_{b_i} = \langle S^{[i+1]} \rangle$, for each $1 \leqslant i \leqslant k$. By induction on i, we can assume that $\langle S^{[i]} \rangle = G^{[i]}$ for some $1 \leqslant i \leqslant k+1$, and it follows that

$$\langle S^{[i+1]} \rangle = \langle S^{[i]} \rangle_{b_i} = G^{[i]}{}_{b_i} = G^{[i+1]}.$$

Moreover, $G^{[k+1]} = \langle S^{[k+1]} \rangle$ is trivial by assumption. □

Now, let us present the Schreier–Sims algorithm, adapted from [50, p. 91], as Algorithm 6. The inputs are a (possibly empty) sequence $B \subset \Omega$ and a generating set S for the group $G \leqslant \mathrm{Sym}(\Omega)$. The algorithm extends both B and S to form a BSGS for G. The first step involves extending B, if required, so that no element of S fixes all points in B, since this is

necessary for B to be a base. Once this is established, we enter the main loop. The algorithm computes the basic orbits, associated transversals, and partial generating sets, with respect to B and S. We define $H^{[i]} = \langle S^{[i]} \rangle$ and notice that, because of the first step, we have $H^{[k+1]} = 1$. The algorithm now proceeds by determining whether $H^{[i]}{}_{b_i} = H^{[i+1]}$ for each decreasing i from k down to 1. By definition, $H^{[i+1]} \leqslant H^{[i]}{}_{b_i}$, so we need only verify that $H^{[i]}{}_{b_i} \leqslant H^{[i+1]}$. This is achieved by testing membership of the generators of $H^{[i]}{}_{b_i}$ within $H^{[i+1]}$. We compute the (Schreier) generators of $H^{[i]}{}_{b_i}$ using the same technique implemented in the orbit-stabiliser algorithm (Algorithm 4), taking care to discard any trivial generators. This is where the the benefit of looping over decreasing i is evident — once the membership test is successful for a particular i, we can be sure the condition is met for all larger i, thus allowing us to apply the Membership algorithm to test for membership in $H^{[i+1]}$.

If the membership test fails for some generator g, we set the Boolean variable y to False, append a new generator h to $S^{[i+1]}$, and extend $H^{[i+1]}$ to a larger subgroup of $G^{[i+1]}$. Note that the new generator h is not necessarily equal to g, but rather the element returned by the Membership algorithm (Algorithm 5) applied to g. If h fixes every element of B, then to maintain the condition $H^{[k+1]} = 1$, we extend B by an element not fixed by h.

4.2 Base size

The most computationally expensive part of the Schreier–Sims algorithm is the multiplication of permutations, in particular if we need to unpack a Schreier vector into a transversal element. In order to improve performance, it would be beneficial to decrease the number of multiplications.

One way to do this is to leverage the observation that every element of a group is determined by its action on a base. Therefore, by representing each element in the Schreier–Sims algorithm by its base image, we can make the cost of multipli-

Algorithm 6 The Schreier–Sims algorithm

Inputs: A group $G \leqslant \mathrm{Sym}(\Omega)$, $B \subset \Omega$ and S, a generating set for G.
Output: A base B; a set of Schreier generators S.

for $g \in S$ **do** ▷ First extend B using S
  if $g \in G_{b_1,\ldots,b_k}$ **then**
    Find $\alpha \in \Omega$ not fixed by g;
    Append α to B;
    $k = k + 1$;
  end if
end for
for $i \in [1, \ldots, k]$ **do**
  $S^{[i]} = S \cap G_{b_1,\ldots,b_{i-1}}$;
  $H^{[i]} = \langle S^{[i]} \rangle$;
  $\Delta^{[i]} = {b_i}^{H^{[i]}}$;
end for
$i = k$;
while $i \geqslant 1$ **do** ▷ Test if $H^{[i+1]} = {H^{[i]}}_{b_i}$
  for $b \in \Delta^{[i]}$ **do**
    Find $u_b \in H^{[i]}$ with ${b_i}^{u_b} = b$;
    for $g \in S^{[i]}$ **do**
      if $u_b g \neq u_{bg}$ **then** ▷ Test if $u_b g u_{bg}^{-1} \in H^{[i+1]}$ is a Schreier generator
        $y :=$ True;
        $h, j :=$ Membership$(u_b g u_{bg}^{-1}, B, S, \Delta^*)$;
        if $j \leqslant k$ **then** ▷ New strong generator h in $G^{[j]}$
          $y :=$ False;
        else if $h \neq 1$ **then** ▷ h fixes all pts in B, so extend B
          $y :=$ False;
          Find $\alpha \in \Omega$ not fixed by h;
          Append α to B;
          $k := k + 1$;
          $S^{[k]} := []$;
        end if
        if $y =$ False **then**
          for $l \in [i + 1, \ldots, j]$ **do**
            Append h to $S^{[l]}$;
            $H^{[l]} := \langle S^{[l]} \rangle$;
            $\Delta^{[l]} := {b_l}^{H^{[l]}}$;
          end for
          $i := j$;
          continue i;
        end if ▷ Now $u_b g u_{bg}^{-1} \in H^{[i+1]}$ is confirmed
      end if
    end for
  end for ▷ Now ${H^{[i]}}_{b_i} = H^{[i+1]}$ is confirmed
  $i := i - 1$;
end while
$S := \bigcup_{i=1}^{k} S^{[i]}$;
return B, S;

cation proportional to the length of a base, and then we can attempt to minimise this length.

This has been one of the motivations for studying base lengths for permutation groups. Let $b(G)$ denote the size of the smallest base of $G \leqslant \mathrm{Sym}(\Omega)$, also known as the *base size* for G. As described above, the base size for a permutation group can impact how efficiently it may be represented. Pyber [87] showed that there is a universal constant $c > 0$ such that almost all subgroups G of S_n have $b(G) > cn$. Moreover, Blaha [9] showed that finding the base size of an arbitrary group G is an NP-complete problem.

Despite this fact, significant research has been conducted into primitive groups, which seem to have small base size relative to their degree. Indeed, Halasi, Liebeck and Maróti [45] proved that, if $G \leqslant \mathrm{Sym}(\Omega)$ is primitive, then

$$b(G) \leqslant 2\log|G|/\log|\Omega| + 24.$$

This result, building on a previous version with non-explicit constants by Duyan, Halasi and Maróti [33], settled a conjecture of Pyber that had been open for 30 years.

It further seems that many families of primitive groups have base size 2. Note that this is the smallest interesting base size, since a primitive group with base size one must necessarily be cyclic of prime order. This motivated Saxl to initiate a programme to classify the primitive groups with base size 2. Many authors have contributed to this classification (e.g., see [17], [16], [36], [37], [38], [57], and [58]), but there is still much work to be done. It should be noted in particular that, so far, the majority of results in this direction are non-constructive.

4.3 New frontiers: the normaliser problem

The *normaliser problem* takes as input (generating sets for) two subgroups $G, H \leqslant S_n$, and asks for generators of $N_G(H)$. Currently, the asymptotically best known algorithm for this prob-

lem is due to a 2020 result by Wiebking [96], which runs in time $2^{O(n)}$. There currently do not exist efficient algorithms to solve the general normaliser problem in practice.

Luks [66] showed that the problems of computing set stabilisers, centralisers and intersections of permutation groups (all members of the so-called *Luks class* of problems) are all polynomial time reducible to the normaliser problem. Moreover, the complexity of algorithms lying in the Luks class was integral to Babai's breakthrough on the graph isomorphism problem [2]. Hence, efficient approaches to the normaliser problem have the potential for significant impact.

Luks and Miyazaki [67] showed in 2011 that the normaliser problem is solvable in polynomial time for groups with restricted composition factors. In 2020, Roney-Dougal and Siccha [88] showed that, if $G = \langle X \rangle$ is a primitive permutation group, then $N_{S_n}(G)$ can be computed in quasipolynomial time, namely $|X| \cdot 2^{O(\log^3 n)}$. This was subsequently extended by Chang and Roney-Dougal [24] to permutation groups H, G whose normalisers in S_n are primitive — namely, one can, in quasipolynomial time, check if $N_{S_n}(H)$ is primitive and if so, compute $N_G(H)$. Further work by Chang, Jefferson and Roney-Dougal [23] considers the normaliser problem for groups with a large number of orbits.

Despite significant progress on the normaliser problem over the past five years, there remains considerable scope for further advancements.

5 Finitely presented groups

Finite presentations offer a compact and natural way to define a group. However, they can also be the most difficult to deal with computationally. We begin by recalling some theory.

Definition 15 The *free group* F_X on a set X is the set X^* of all

words on $X \cup X^{-1}$, along with the binary operation of concatenation.

Definition 16 For a set X and $R \subseteq X^*$, the (*group*) *presentation* $\langle X \mid R \rangle$ is defined to be the quotient group F_X/N, where N is the normal closure of R in F_X.

A *finitely presented group* is a group defined by a presentation $\langle X \mid R \rangle$ where both X and R are finite.

Intuitively, the group defined by the presentation $\langle X \mid R \rangle$ is the largest group on X where all words in R are equal to the identity.

Many fundamental questions about finitely presented groups are undecidable in general. Indeed, Novikov [80], Boone [10] and Britton [15] independently proved that one cannot decide the finiteness of a finitely presented group in general, nor solve the word problem (i.e., determining when a word is equal to the identity).

Because of the importance of these groups, the development of practical algorithms for investigating them has been a focus of research. A substantial body of work includes methods that are motivated by the following theorem:

Theorem 17 (von Dyck's theorem) *Let* $G = \langle X \mid R \rangle$, *and consider a map* $\alpha : X \to H$ *into a group* H. *If every relator* $x_1^{e_1} \dots x_r^{e_r} \in R$ *satisfies*

$$\alpha(x_1)^{e_1} \dots \alpha(x_r)^{e_r} = 1_H,$$

then α *extends to a group homomorphism* $G \to H$.

In other words, if we can find generators for H that satisfy the relators of G, then H is a quotient of G by the first isomorphism theorem for groups.

As a consequence of von Dyck's theorem, considerable research is dedicated to the development of *quotient algorithms*, which attempt to construct quotients of finitely presented groups. It is not always guaranteed that we can find a finite quotient. Indeed, Bridson and Wilton showed in 2015 that one cannot

even design an algorithm to decide whether a finitely presented group has a proper subgroup of finite index (see [14]).

Despite this setback, quotient algorithms are still widely used in practice. Epimorphisms from finitely presented groups onto types of groups that are easier to compute with, such as permutation groups, are particularly valuable, as they enable practical computations within the quotient, allowing us to answer questions about the finitely presented group that are undecidable in general.

The Todd-Coxter algorithm for coset enumeration is one of the earliest examples of such an algorithm, published in 1936 (see [94]) and still widely used today. We discuss this algorithm in detail in Section 5.4, after we have outlined some other important approaches.

5.1 Abelian invariants

Recall that, if G is a finitely presented group, then the factor group $G/[G, G]$ is the largest abelian quotient of G, also called its *abelianisation* and denoted as G^{ab}. By the fundamental theorem of finitely generated abelian groups, G^{ab} is isomorphic to a direct product $\mathbb{Z}^n \times C_{t_1} \times C_{t_2} \times \ldots \times C_{t_m}$, where each t_i is a prime power. The *abelian invariants* of G are then the numbers $t_1, t_2, \ldots t_m$, arranged smallest to largest, along with n zeros.

Identifying the isomorphism type of G^{ab} through calculating its abelian invariants can be sufficient to address key questions about G. For example, if G^{ab} contains $\mathbb{Z}$ as a subgroup, then G must be infinite. This condition, however, is clearly not necessary.

The following lemma demonstrates that it is easy to find a presentation for the abelianisation of a finitely presented group.

Lemma 18 *Let* $G = \langle X \mid R \rangle$ *be a finitely presented group. Then*

$$G/[G, G] \simeq \langle X \mid R \cup [X, X] \rangle,$$

where $[X, X] = \{[x, y] \mid x, y \in X\}$.

Proof — By definition, G is the quotient of the free group F_X on X by the normal closure $N = \langle R^{F_X} \rangle$. Moreover,

$$\langle X \mid R \cup [X, X] \rangle \simeq F_X / \widetilde{N},$$

where $\widetilde{N} = \langle (R \cup [X, X])^{F_X} \rangle$. Every pair of generators commute in $F_X/\widetilde{N}$, so $F_X/\widetilde{N}$ is abelian. Now, $(F_X/N)/(\widetilde{N}/N) \simeq F_X/\widetilde{N}$ is an abelian quotient of G by the third isomorphism theorem, so $[G, G] \leqslant (\widetilde{N}/N)$. In addition,

$$\widetilde{N} = \langle (R \cup [X, X])^{F_X} \rangle = \langle R^{F_X} \cup [X, X]^{F_X} \rangle = NM,$$

where $M = \langle [X, X]^{F_X} \rangle$. Hence, $\widetilde{N}/N = [G, G]$, as required. □

Hence, computing the abelian invariants of a finitely presented group may be reduced to the problem of determining the structure of abelian finitely presented groups. This can be achieved using a number of approaches; many involve computing the *Hermite* or *Smith normal form* of a certain matrix defined from G^{ab}. We direct the reader to [50, Section 9.2] for more information on the techniques and related computational challenges.

5.2 Polycyclic groups

A group G is said to be polycyclic if there exists a chain of subgroups (called a *polycyclic series*)

$$G = G_1 > G_2 > \ldots > G_n = 1$$

where, for each $1 \leqslant i \leqslant n-1$, G_{i+1} is normal in G_i and G_i/G_{i+1} is cyclic. The number of infinite cyclic quotients in this series is called the group's *Hirsch length*.

Clearly, every polycyclic group is solvable, and polycyclic groups may be characterised as being precisely the solvable groups in which every subgroup is finitely generated. In partic-

ular, every finite solvable group is polycyclic, as are all finitely generated nilpotent groups.

Polycyclic groups form an important class from a computational group theory standpoint. In addition to being closed under taking subgroups, quotients and extensions, they can also be stored efficiently via *polycyclic presentations*. Moreover, some problems that are undecidable for arbitrary finitely presented groups, such as the word problem, may effectively be solved for groups in this family.

Definition 19 A presentation $\langle X \mid R \rangle$ of a polycyclic group G is said to be a *polycyclic presentation* if it consists of:

(1) *Polycyclic generating sequence*: A generating set

$$X = (x_1, x_2, \ldots, x_n),$$

where $G_i = \langle x_i, G_{i+1} \rangle$, for each $1 \leqslant i \leqslant n-1$.

(2) *Power relations*: If $|G_i : G_{i+1}| = p_i$ is finite, then ${x_i}^{p_i}$ can be written as a word in $x_{i+1}, \ldots, x_n$.

(3) *Conjugacy relations*: If $i < j$, then ${x_j}^{x_i}$ and ${x_j}^{x_i^{-1}}$ can be written as a word in $x_{i+1}, \ldots, x_n$.

By convention, conjugacy relations of generators that commute are usually omitted from a polycyclic presentation. For this reason, we distinguish polycyclic presentations from regular group presentations by writing $G = \mathrm{Pc}\langle X \mid R \rangle$, notation which is adopted in both GAP and Magma. A polycyclic group given as a group defined by a polycyclic presentation is often called a *PC-group*. Even with the omission of so-called trivial polycyclic relations, the relations in a polycyclic presentation are usually quite redundant, but their format is useful for computational purposes.

For the rest of this subsection, we let G be a PC-group, and adopt the notation given in Definition 19.

Using a method similar to that employed in the proof of Lemma 13 in Section 4, one may show that every element $g \in G$ may be written uniquely as a product

$$x_1^{e_1} x_2^{e_2} \dots x_n^{e_n},$$

with each e_i such that $1 \leqslant e_i \leqslant p_i$ if p_i is finite. This is called the *normal form* of g. The normal form of elements in G may be computed using a *collection algorithm*, which leverages the polycyclic presentation. Crucially, according to [50, Theorem 8.15], a collection algorithm applied to an element of a PC-group is guaranteed to terminate, and this fact implies that the word problem is decidable for these groups. The first implementation of a collection algorithm was carried out by Neubüser in the early 1960s (see [71]), with later refinements by several authors, for example [41], [47], and [62].

PC-groups are also amenable to a range of effective algorithms to compute subgroups, homomorphisms and quotients, orbits and stabilisers on a finite set, and complements and extensions. Moreover, in addition to the word problem, conjugacy and isomorphism are decidable for PC-groups, making the existence of a polycyclic presentation for a finitely presented group highly desirable. See [50, Chapter 8] for more details.

5.3 Solvable quotients

Following a similar approach as the computation of abelian invariants, solvable quotient algorithms aim to construct an epimorphism from a finitely presented group onto as large a solvable group as possible, so that effective algorithms may be carried out in the quotient.

Many of the developments in this direction have been oriented towards p-group quotients. One of the most prominent results of this type is the well-known algorithm of Macdonald [68], refined by Newman and O'Brien [75], and Havas and Newman [46]. This algorithm attempts to construct, for an input

prime p, the largest quotient of G that is a finite p-group. It is not uncommon for a largest such quotient to not exist, so the user is also able to specify a bound on the nilpotency class of the p-quotient one would like to construct. A generalisation to nilpotent quotients was developed by Nickel in 1994 (see [76]), and is still used in Magma and GAP today.

There are several algorithms for constructing solvable quotients of finitely presented groups as iterated extensions, including those developed by Leedham-Green [60], Niemeyer [77], and Plesken [85].

Lo [64] proposed an algorithm in the late 1990s to obtain polycyclic quotients of finitely presented groups using Gröbner bases. A refinement and generalisation was developed in 2010 by Eick and Horn [34].

5.4 The Todd–Coxeter algorithm

In this section, we describe the Todd–Coxeter algorithm for coset enumeration, following [50, Section 5.1]. Let us first fix some notation that we will maintain throughout the section. Let $G = \langle X \mid R \rangle$ be a finitely presented group, where

$$X = \{x_1, x_2, \ldots, x_s\} \quad \text{and} \quad R = \{r_1, r_2, \ldots, r_t\}.$$

Let $H \leqslant G$ be the subgroup generated by elements $\{y_1, y_2, \ldots, y_k\}$, where each y_i is given as a word in $X \cup X^{-1}$.

Data structures. The Todd–Coxeter algorithm requires three key data structures, each represented as a table or a set of tables that are progressively completed. We will now introduce these data structures, along with an explanation of how they are labelled and populated.

1. **Coset table:** M
    - The rows of M are labelled by positive integers corresponding to cosets of H.

- The columns of M are labelled by $X \cup X^{-1}$.
- The (k, g)-entry of M is the positive integer ℓ corresponding to the coset kg.
- We initialise M by starting with one row labelled 1, corresponding to H.

2. **Relator tables:** M_i, for each relator $r_i = x_{i_1} x_{i_2} \dots x_{i_m} \in R$.
   - The rows of M_i are labelled by positive integers corresponding to cosets of H (as in M).
   - The columns of M_i are labelled by $x_{i_1}, x_{i_2}, \dots, x_{i_m}$.
   - The (k, x_{i_j})-entry of M_i is the positive integer ℓ corresponding to the coset $kx_{i_1} x_{i_2} \dots x_{i_j}$.
   - Each time we add a row, labelled with a positive integer d, to M, we add a corresponding row to M_i and fill in the (d, x_{i_m})-entry of M_i with d.
3. **Subgroup tables:** S_j, for each generator $y_j = x_{j_1} x_{j_2} \dots x_{j_t}$.
   - Each subgroup table S_j has a single row corresponding to the coset H.
   - The columns of S_j are labelled by $x_{j_1}, x_{j_2}, \dots, x_{j_t}$.
   - The $(1, x_{j_a})$-entry of S_j is the positive integer ℓ corresponding to the coset $1x_{j_1} x_{j_2} \dots x_{j_a}$.
   - We initialise S_j by setting the $(1, x_{j_t})$-entry to 1, since we have that $Hy_j = H$.

Process. The Todd–Coxeter algorithm proceeds as follows:

1. **Initialisation**: Initialise the coset, relator and subgroup tables as described above.
2. **New coset definition**: Choose the smallest d, i such that there is no entry in row d and column x_i of the coset table.

Write d' there, where d' is the smallest coset label not already used. We now assume x_i maps coset d to d' and x_i^{-1} maps d' to d.

3. **Fill relations**: Using the information from Step 2 or an earlier iteration of this step, update the relator and subgroup tables by applying any deductions. For example, to incorporate the information $dx_i = d'$, scan each row of these tables for an entry d. For each such entry d, check if the next column is labelled x_i. If so, verify whether the corresponding entry in that row and column is already filled. If it is empty, assign it the value d'. If it already contains d', leave it unchanged. However, if it contains a different entry, a *collision* has occurred, so transition to Step 4. If there is no row for d' in the coset table, no collisions have occurred and all tables have been scanned and updated, add d' as a new row in the coset and relator tables. Initialise these rows as described in Step 1, and populate any entries using the information already obtained.

4. **Collision handling**: If a collision occurs between cosets with labels d', d'' with $d'' < d'$, then merge them by replacing every instance of d' with d'', and propagate this identification throughout the tables, ensuring consistency. Resume Step 3 with this new labelling.

5. **Iteration:** Repeat Steps 2 and 3 until every entry of the coset, subgroup and relator tables is completed.

6. **Termination**: The algorithm terminates when all table entries are filled, or no new coset labels can be added. The completed coset table represents the action of G on the cosets of H, and this table, along with the index of H in G (the number of rows in the coset table) is returned.

Application. We now present an illustative example of an application of the Todd–Coxeter algorithm.

Example 20 We apply the Todd–Coxeter algorithm to the finitely presented group $G = \langle a, b \mid ab^{-1}ab, b^2 \rangle$, isomorphic to the infinite dihedral group, and the subgroup $H = \langle a^2 \rangle$. We initialise the tables as described above, and then we start the algorithm, introducing new cosets in the coset table as necessary, from top-left to bottom-right. The first ten steps of the algorithm are described in Table 1, where each of the bolded entries in the last column indicates the place where a deduction arises for the next step of the algorithm.

Step	Type	New Information	New entries
0	—	Initialisation	$M(1,\ )$, $S_1(1, a^2)$, $M_1(1, ab^{-1}ab)$, $M_2(1, b^2)$
1	Define	$1a = 2$	$M(1, a)$, $\mathbf{S_1(1, a)}$, $M_1(1, a)$
2	Fill	$2a = 1$	$M(1, a^{-1})$
3	Fill	Add coset 2	$M(2, a^{\pm 1})$, $M_1(2, a)$, $M_1(2, ab^{-1}ab)$, $M_2(2, b^2)$
4	Define	$1b = 3$	$M(1, b)$, $\mathbf{M_2(1, b)}$
5	Fill	$3b = 1$	$M(1, b^{-1})$, $M_1(2, ab^{-1})$
6	Fill	Add coset 3	$M(3, b^{\pm 1})$, $M_1(3, ab^{-1}ab)$, $M_2(3, b^2)$, $M_2(3, b)$
7	Define	$2b = 4$	$M(2, b)$, $M_2(2, b)$
8	Fill	Add coset 4	$M(4, b^{-1})$, $M_1(4, ab^{-1}ab)$, $M_2(4, b^2)$, $\mathbf{M_2(4, b)}$
9	Fill	$4b = 2$	$M(2, b^{-1})$, $M(4, b)$, $M_1(1, ab^{-1})$, $M_2(4, b)$
10	Define	$3a = 5$	$M(3, a)$, $M_1(2, ab^{-1}a)$, $M_1(3, a)$

Table 1: The first ten steps of the Todd–Coxeter algorithm for the infinite dihedral group, with bolded entries indicating sources of new deductions.

Note that, for ease in table indexing, we replace the standard

labels of the columns in the subgroup and coset tables with the corresponding partial word in the generators of the group. In the eleventh step we deduce from entry $M_1(2, ab^{-1}a)$ that $5b = 2$. However, when attempting to propagate this deduction through to the coset table, we see that the $M(2, b^{-1})$ entry is already filled with the coset labelled 4. Hence, we have a collision, and must replace every instance of the coset labelled 5 with the coset labelled with 4. We conclude the example here, having illustrated the process of the algorithm. The completed tables achieved at the conclusion of the algorithm appear in Figure 1.

Coset	a	b	a^{-1}	b^{-1}
$\underline{1}$	2	3	2	3
2	1	4	1	4
3	4	1	4	1
4	3	2	3	2

(a) Coset table M

Coset	a	a^2
$\underline{1}$	2	$\underline{1}$

(b) Subgroup table S_1

Coset	a	ab^{-1}	$ab^{-1}a$	$ab^{-1}ab$
$\underline{1}$	2	4	3	$\underline{1}$
2	1	3	4	2
3	4	2	1	3
4	3	1	2	4

(c) Relator table M_1

Coset	b	b^2
$\underline{1}$	3	$\underline{1}$
2	4	2
3	1	3
4	2	4

(d) Relator table M_2

Figure 1: The tables for the Todd–Coxeter algorithm applied to the infinite dihedral group $G = \langle a, b \mid ab^{-1}ab, b^2 \rangle$, with starting entries underlined.

The fact that the Todd–Coxeter algorithm does indeed return the index of H in G, along with a permutation representation, is not completely trivial; a proof may be found in Chapter 5 of [50], along with several other results making aspects of the Todd–Coxeter algorithm mathematically rigorous.

There exist a number of variations on the Todd–Coxeter algo-

rithm, which have different choices of new coset placement and levels of deduction per step. We generally want to maximise deductions to avoid collisions, since these can be very computationally expensive to rectify. There are three implementations that have found extensive use.

The method proposed by Felsch [39] closely resembles the original manual technique by Todd and Coxeter. It aims to reduce redundant cosets (i.e., collisions) by checking for coincidences from all cyclic permutations of relators each time a coset is introduced. The HLT (Haselgrove–Leech–Trotter) method processes each coset sequentially, applying all relations to it and introducing any necessary new cosets to facilitate this (see [95] and [59]). The "lookahead" algorithm, first implemented by Guy [44], instead alternates between two phases. In the "defining" phase, it functions like the HLT method, while in the "lookahead" phase it applies all relations to the currently defined cosets, to identify coincidences that can be derived without the need to introduce new cosets. More information about the differences of these methods in theory and practice can be found in [20].

Having a permutation representation of G acting on cosets of H makes it possible to deduce various properties of G. For example, given elements w, w' of G as words in the generators, one can:

- decide membership of w in H by testing whether $Hw = H$;
- obtain a negative answer to the word problem: namely that $w \neq w'$ if $Hw \neq Hw'$.

Notice that the application to the word problem only gives a sufficient but not necessary condition.

In addition to these applications of the permutation representation of G, obtained from the Todd–Coxeter algorithm, the ability to perform coset enumeration for G on $H\backslash G$ is crucial for other algorithms that deal with finitely presented groups.

For example, let us consider the problem of trying to find all subgroups $H < G$ of some finitely presented group G, up to some index N. The first practical algorithms to solve this problem were constructed independently by Sims [92] and Dietze and Schaps [32] in 1974. Both involve performing coset enumeration of the identity subgroup in G, but only allowing at most $f(N)$ cosets to be defined, where $f(N) \geqslant N$, and usually $f(N) \leqslant 2N$. As the coset enumeration progresses, we systematically force collisions between cosets, and then attempt to complete the coset enumeration with these extra constraints. If the coset enumeration completes successfully without introducing more than N cosets, then we have found a subgroup H of G with index at most N. This approach was implemented in CAYLEY (a precursor to MAGMA) as well as GAP. An alternative approach, proposed by Sims [93] in the 1990s, instead uses a backtrack search to attempt to fill in incomplete coset tables, rather than forcing collisions. This technique is more efficient in many situations, and as a result it underlies the current low index subgroup algorithms in MAGMA.

Other algorithms for finitely presented groups, that rely on either coset enumeration or the existence of a coset table for $H < G$, include the Reidemeister–Schreier algorithm for subgroup presentations, rewriting algorithms for subgroup presentations (see [50, Section 5.3]), and even the "reverse Todd–Coxeter algorithm" for finding presentations for permutation groups [19].

5.5 New frontiers

Algorithms for finitely presented groups continue to be a broad and active area of research in computational group theory. Here we mention two particular directions: homomorphisms of finitely presented groups onto insoluble groups, and the use of polycyclic groups in cryptography.

Homomorphisms onto insoluble groups. One consequence of the undecidability of whether a finitely presented group has a non-trivial finite index subgroup is that there is also no algorithm to determine all finite simple images of a finitely presented group. Despite this, work by Plesken and Fabiańska [86], later refined by Jambor [55], shows that there exists an algorithm to determine all images of a finitely presented group that are isomorphic to either $\mathrm{PSL}_2(q)$ or $\mathrm{PGL}_2(q)$ and, if the number of possible choices for q is finite, compute a complete list. A similar algorithm was later developed by Jambor [54] to deal with 2-generated finitely presented groups with a quotient isomorphic to either $\mathrm{PSL}_3(q)$, $\mathrm{PGL}_3(q)$, $\mathrm{PSU}_3(q)$ or $\mathrm{PGU}_3(q)$. In [13] the authors investigated the existence of homomorphisms onto infinite families of simple groups more broadly. They show that, if the infinite family $\mathcal{F}$ of simple groups consists of groups of Lie type of fixed dimension, then there exist analogous algorithms to those described above. On the other hand, if $\mathcal{F}$ contains infinitely many alternating groups, or classical groups of infinitely many dimensions, the same authors show that no such algorithm exists.

There are few known approaches to extending insoluble quotients to larger ones. One strategy in the context of perfect groups was proposed by Holt and Plesken [51] in the late 1980s, while an alternative was proposed by Dietrich and Hulpke in 2021 (see [28]). Both extend techniques used in iterative quotient algorithms for soluble groups.

There is a large scope for further work in this area, including a practical implementation of the results of [13], and further approaches for extending simple quotients.

Polycyclic groups in cryptography. Polycyclic groups have found application in *group-based cryptography*, beginning with work of Eick and Kahrobaei in 2004 (see [35]). One genre of protocol in this field is based on the complexity of the conjugacy search problem — namely, given conjugate elements g, h in a group G,

find $x \in G$ with $g^x = h$. Note that this is a special case of the conjugacy problem which, as mentioned in Section 5.2, is decidable for polycyclic groups. Despite being decidable, Eick and Kahrobaei showed that the time complexity of the conjugacy problem for polycyclic groups is exponential in the Hirsch length, while the word problem remains efficiently solvable. This combination makes polycyclic groups suitable for use in cryptography, and there has been considerable research activity in this area over the past 20 years. See [43] for a survey, along with a number of open problems.

6 Matrix groups

Despite their importance in a range of fields, matrix groups present significant computational challenges. The Handbook of Computational Group Theory (see [50, Section 7.8]) itself said "It is frequently impossible to study the structure of a small matrix group of relatively small dimension". This is not only due to the size of these groups and their related vector spaces. Just as with finitely presented groups, there are also decidability considerations for infinite groups. For example, membership in $G \subseteq GL_4(\mathbb{Z})$ is undecidable, let alone considering higher dimensions.

In this section, we will mostly assume our matrix groups are over finite fields $\mathbb{F}_q$.

An early approach to computing with matrix groups was based on the observation that a matrix group $G \leqslant GL(V)$ acts faithfully on the underlying vector space V. Thus, we may instead consider G as a permutation group $G' \leqslant \mathrm{Sym}(V)$ of the vectors in V, and use algorithms for permutation groups to work with G. This approach is only feasible for matrix groups and vector spaces of modest size, since the input size of each matrix is now q^n, rather than $n^2 \log_2 q$ when represented as a matrix.

There were also attempts to instead generalise the base and strong generating set techniques, used for permutation groups, to a more general setting. The first version of the Schreier–Sims algorithm for matrix groups was constructed by Butler [18]. As we saw in Section 4, the basic orbits $\Delta^{[i]} = v_i^{G^{[i]}}$ play a fundamental role in the efficiency of algorithms which rely on BSGS. Since the degree q^d of G' is exponential in d, basic orbits become large very quickly, so alternative strategies which produce smaller basic orbits seem essential. In particular, Cannon et al. [21] commented in a 2019 paper that basic orbits of size "up to 10^4 is desirable, but computation remains feasible with ... lengths exceeding 10^6".

Butler's approach to reducing the sizes of basic orbits involves mapping a matrix group G to a permutation group of smaller degree than $|V|$. Specifically, he considered the action of matrix groups on one-dimensional subspaces of the underlying vector space. Since this action is not always faithful, Butler augmented the set of subspaces with a selection of vectors to achieve faithfulness. In contrast, Murray and O'Brien [70] adopted a different strategy, focusing on carefully selecting base points, within V, which are expected *a priori* to have small orbits. Their method computes the common eigenspaces of random group elements, trying to ensure that the stabiliser generated by these elements is sufficiently large. Their algorithm is implemented in Magma.

Despite these developments, limitations of computational resources for storing and working with large matrix groups remained a problem. This led to the establishment of the Matrix Group Recognition Project (MGRP). This programme was established in the early 1990s (see [61]) with the aim to produce efficient algorithms for solving problems with matrix groups over $\mathbb{F}_q$.

One of the core features of most algorithms for matrix groups is that they are based on random sampling of elements. Two genres of such algorithms for decision (yes/no) problems for

matrix groups prevail. A *Monte Carlo algorithm* is one that returns an incorrect answer with chosen probability, less than a half. If the algorithm is always correct when it returns a particular answer (either true or false), then it is said to be *one-sided*. In contrast, a *Las Vegas* algorithm never returns an incorrect answer, but instead returns fail with a prescribed probability. In both cases, the chosen error probability can be adjusted by altering the number of random trials conducted.

Moreover, many algorithms require selection of group elements uniformly at random. One of the most famous algorithms for computing random elements from a set of generators is the product replacement algorithm, developed in [22].

There have been two main strategies to realise the aims of the MGRP. The first, the so-called *black-box approach*, seeks to build a certain characteristic series of subgroups for G which can then be refined into a chief series. The algorithms involved are independent of the group's specific representation, so they may be used in a broader context than just matrix groups.

The *geometric approach*, instead, leverages the natural linear action of $G \leqslant \mathrm{GL}(V)$ on its vector space V to construct a composition series. Here, the linear representation of G is integral to the algorithms we apply. This approach also incorporates some black box algorithms to identify almost simple groups.

Common to both of these approaches is a reliance on the ability to solve three major problems for a (quasi)simple matrix group $G \leqslant \mathrm{GL}(V)$:

1. *Naming:* Determine the name/identity of G.

2. *Constructive recognition:* Construct an isomorphism (possibly modulo scalars) to a "standard copy" of G.

3. *Constructive membership:* Let $x \in \mathrm{GL}(V)$. Determine whether $x \in G$. If so, write x as a straight line program in the generators of G.

6.1 The black-box approach

The notion of a black box group was first introduced by Babai and Szemerédi in 1984 (see [6]). Rather than defining a class of groups, the term "black box group" instead refers to a particular method of representating a group on a computer. Each element of a black box group G is given as a string of bits of length at most a fixed constant N. Given two words representing elements $g, h \in G$, there exist oracles (that is, abstract programmes that provide a correct answer to a problem in constant time) which can compute gh, g^{-1}, and determine whether $g = h$. These constitute the only group operations permissible in black box groups. Since the length of words representing group elements is bounded by a constant N, a black box group is necessarily finite. Finite permutation groups and matrix groups over finite fields are important examples of black box groups (note that the algorithms for composition, inversion, and equality testing are oracles, and hence do not need to be implementable here).

Algorithms for black box groups are desirable because they may be used in a broad range of settings, independent of the type of representation of a group. Time and space complexity analysis also benefits from the bit string encodings and constant time oracles. However, a major disadvantage of black box algorithms is that they cannot exploit any underlying structure present in other representation types — for example, the computation of orbits and stabilisers, which are essential in algorithms for permutation groups.

The black box approach towards the Matrix Group Recognition Project was started and led by Babai and Beals — a comprehensive survey is given in [3]. A key breakthrough in this approach was made in 2009 by Babai, Beals and Seress [5], building on 25 years of work by various authors. They prove that if $G \leqslant GL_d(q)$ then, assuming the existence of oracles for discrete logarithm and for the factorisation of integers of the

form $q^i - 1$, for $1 \leqslant i \leqslant d$, there exist polynomial time Monte Carlo black box algorithms that construct a certain characteristic series for G when q is odd, identify its composition factors, and solve the constructive membership problem for G. If instead q is even, they are able to compute a composition series for $G/\mathrm{Rad}(G)$, where $\mathrm{Rad}(G)$ is the solvable radical of G, as well as name its composition factors.

In the past decade, there has been further work on black box algorithms to solve the constructive recognition problem for groups of Lie type (see, for example, [31] and [56]).

6.2 The geometric approach

In contrast to the abstraction of the black box approach, the geometric approach to the MGRP leverages the representation of a matrix group G to deduce its structure. Namely, the aim is to construct a composition series for G by exploiting the geometric properties of G acting on its underlying vector space.

In order to construct this composition series for G, we rely on Aschbacher's theorem, whose classification of the maximal subgroups of $\mathrm{GL}_n(q)$ is crucial in this approach. The following is a paraphrased version.

Theorem 21 *Let V be a finite d-dimensional vector space over $\mathbb{F}_q$ and suppose $G \leqslant \mathrm{GL}_d(q)$ is maximal, with centre $Z = Z(G)$. Then one of the following holds*:

C1. *G acts reducibly on V.*

C2. *G preserves a direct sum decomposition $V_1 \oplus V_2 \oplus \ldots \oplus V_t$ of V into $t > 1$ subspaces, each of dimension $s = d/t$. Moreover, G permutes these subspaces transitively, so $G \leqslant \mathrm{GL}_s(q) \wr \mathrm{Sym}(t)$.*

C3. *For some $e \mid d$ with $e > 1$, G embeds in $\Gamma\mathrm{L}_{d/e}(q^e)$ and acts on V.*

C4. *G preserves a tensor decomposition $V_1 \otimes V_2$ of V, with tensor factors of dimensions $d_1, d_2 > 1$, and G is contained in the central product of $\mathrm{GL}_{d_1}(q)$ and $\mathrm{GL}_{d_2}(q)$.*

C5. G *can be defined modulo scalars over a proper subfield of* $\mathbb{F}_q$. *In other words, some* $\mathrm{GL}_d(q)$*-conjugate of* G *lies in* $\mathrm{GL}_d(q_0)$, *where* $\mathbb{F}_{q_0}$ *is a proper subfield of* $\mathbb{F}_q$.

C6. *There is a prime* r *and integer* n *such that* $d = r^n$, *and* G *normalises an extraspecial group of order* r^{2n+1}, *or* G *is symplectic type and normalises a group of order* 2^{2n+2}.

C7. G *is tensor-induced. Namely,* G *preserved a tensor product decomposition* $V_1 \otimes V_2 \otimes \ldots \otimes V_t$ *of* V *into subspaces of dimension* $s = d^{1/t}$. *Moreover,* G *permutes the set of tensor factors transitively, hence* $G/Z \leqslant \mathrm{PGL}_s(q) \wr \mathrm{Sym}(t)$.

C8. G *is a subgroup of the normaliser of a classical group in its natural representation.*

C9. G/Z *is almost simple.*

Notice that Aschbacher classes are in general not mutually exclusive.

Aschbacher's theorem implies that, except for groups falling only into C9, which can be dealt with separately, every matrix group preserves some natural structure in its action on the underlying vector space. Often, this implies the existence of a normal subgroup $N \lhd G$. Therefore, a natural first step in trying to identify and decompose a given matrix group is to determine (at least one of) its Aschbacher classes. If this is successful, we can then use specialised algorithms designed to deal with groups in this class, and usually reduce the size of the problem. For example, if a proper normal subgroup N of G can be constructed, then we instead conduct analysis on N and G/N. In particular, we may be able to construct further normal subgroups of N and G/N, and work towards a composition series for G.

We do not give here a comprehensive overview of algorithms that decide membership in particular Aschbacher classes — we refer the reader instead to the excellent survey article [81, Section 4] on the subject. As an example, we discuss the MeatAxe,

used to identify and deal with reducible matrix groups falling into class C1.

Reducible subgroups and the MeatAxe. The MeatAxe was introduced by Parker [83] in the early 1980s. It is a Las Vegas algorithm developed to determine if a matrix group $G \leqslant GL(V)$ acts irreducibly on V, where V is a finite dimensional vector space over a finite field F. Holt and Rees [52] generalised Parker's algorithm by having its efficiency less dependent on field size, and also conducted some complexity analysis. Further improvements and the corresponding complexity analysis were made by Ivanyos and Lux [53].

We now briefly describe the ideas behind the algorithm. Consider V as an FG-module, and let A be the F-algebra generated by the elements of G. Choose an element $a \in A$, then compute and factorise its characteristic polynomial $p(x)$. Let $f(x)$ be an irreducible factor of $p(x)$; then the matrix $f(a) \in A$ has nontrivial nullspace W. If the dimension of W is equal to the degree of $f(x)$, then a short argument shows that V must be irreducible. Otherwise, if $f(x)$ has multiplicity one in $p(x)$, then W is an irreducible $F\langle a\rangle$-module. In this case, we choose a non-zero vector $w \in W$ and compute a basis for the FG-submodule of V generated by w and its images under the generators of G. If this is a proper submodule of V, then G acts reducibly on V. If not, then we repeat the process with a new a chosen at random.

The MeatAxe has complexity $O(d^{7/2} \log q)$ (see [52] and [53]) and works very efficiently on a range of large groups and modules in hundreds of dimensions. Variations and extensions of the MeatAxe are also used to identify groups belonging to other Aschbacher classes. Implementations of the MeatAxe are available in GAP and Magma.

6.3 Naming matrix groups

As mentioned earlier, the goal of the geometric approach is to exploit the structure preserved by a matrix group to determine

its composition series. If we are able to obtain the composition series successfully, we still need to identify the composition factors of the group. There are two categories of naming algorithms for such groups. The first consists of *non-constructive* algorithms, which use data about distributions of elements in a group in order to try to determine its identity. On the other hand, *constructive recognition algorithms* aim to determine not only the identity of the matrix group G, but also an explicit homomorphism from G to a "nice" copy (perhaps a natural representation, or as a permutation group), a way to write arbitrary elements of G as words in the generators.

One of the first developments in the area was an influential 1992 paper by Neumann and Praeger [72], who described a non-constructive one-sided Monte Carlo algorithm for recognising whether a matrix group contains the special linear group in its natural representation. Their work spurred significant interest in a geometric approach to MGRP. The algorithm was later extended to recognition algorithms for other classical groups in their natural representations by Niemeyer and Praeger, in the late 1990s (see [78] and [79]).

Black-box algorithms for (non-constructively) recognising alternating and symmetric groups were firstly developed by Bratus and Pak in 2000 (see [12]) and also in 2002 (see [8]). On the other hand, if a given matrix group is known to be a sporadic simple group, then it can usually be easily identified by the orders of a small number of random elements.

Many of the approaches to constructive recognition of (quasi-)simple groups rely on the notion of *standard generators* introduced by Wilson [98]. While not explicitly defined there, *standard generators* for a group are considered to be described by properties that make them easy to find from a copy of the group given with arbitrary generators, and also provide an obvious way to construct an isomorphism from one copy of a group to another. We illustrate this with an example, a version of which is included as in [98, Example 2].

Example 22 Let $G = Ru$ be the Rudvalis group, a sporadic simple group. This group has order around 2^{37}, and can be generated by elements g_2, g_4 of orders 2, 4 and lying in classes labelled (according to the ATLAS) 2B, 4A, respectively, such that $g_2 g_4$ has order 13.

One may obtain an involution in class 2B as a power of any element of order 26 or 14, and these, along with the 2B elements themselves constitute 16.5% of the group. Similarly, any element $g \in G$ of order 24 has $g^6 \in$4A, and so we can construct an element in 4A with probability around 10.7%.

Any pair of elements lying in 2B, 4A respectively has approximately a 1 in 163 chance of being conjugate to (g_2, g_4), and such pairs are characterised by whether their product has order 13. Hence, we see that (g_2, g_4) are a good choice of standard generators for Ru, since for any copy of Ru given in an arbitrary representation (where we are easily able to compute products and orders of elements), we can easily find these "standard generators", and a map between copies of Ru via standard generators will extend to an isomorphism of groups.

The Composition Tree algorithm. As alluded to in the discussion of Aschbacher's theorem, identifying the geometric structure preserved by a matrix group can afford a method of decomposing it via a homomorphism into a pair of smaller groups.

The iteration of this technique underpins the idea of a *composition tree*, first introduced by Leedham-Green in 2001 (see [61]). In brief, a composition tree is a data structure used to store the composition factors for a group, along with associated data that allow for efficient computation.

Explicitly, a composition tree for a matrix group G is a labelled rooted binary tree. Each node is labelled by a section of G, while the root node corresponds to G. Every non-leaf node labelled with a section H has a left descendant corresponding to a normal subgroup N of H, and a right descendant corresponding to H/N. In particular, the descendants arise as the

kernel and image of a homomorphism. This homomorphism is usually derived from the Aschbacher class of H (if it can be determined), but can also arise in other ways. If there are no non-trivial homomorphisms that can be constructed from the group associated to a node, then such group must be simple. Hence, every leaf of the composition tree corresponds to a composition factor of G.

A prototype of an algorithm to compute a composition tree was first implemented by Leedham-Green and O'Brien in 2000, and a number of variations and refinements were introduced in [74], [82], [81]. In particular, [74] also considers composition tree algorithms for permutation groups that are not amenable to base and strong generating set algorithms, and for groups represented in other manners; an implementation is available in the `recog` package distributed with GAP [73].

A major breakthrough occurred in 2014, when in [1] the authors constructed a highly efficient version of the composition tree algorithm, along with an implementation in Magma [11] as the LMG (Large Matrix Group) suite of functions. This implementation allows the efficient construction of a composition tree for matrix groups, and includes tools for recognizing and manipulating the group's structure.

Explicitly, the authors provide algorithms to construct a composition tree for a matrix group $G \leqslant GL_n(q)$, and then compute:

1. A composition series $1 \leqslant G_1 \leqslant \ldots \leqslant G_m = G$ and, for each k, a subset of G_k that generates G_k modulo G_{k-1}.

2. For each $k > 0$, an effective epimorphism

$$\varphi_k : G_k \to \Sigma_k$$

   where Σ_k is the so-called standard copy of the quotient group G_k/G_{k-1} and $\ker(\varphi_k) = G_{k-1}$.

3. For each $k > 0$, a *rewriting algorithm* for Σ_k that allows us to represent each element in a normal form.

4. For each $k > 0$, a map $\Phi_k : \Sigma_k \to G_k$ such that

$$\Phi_k \circ \varphi_k = \mathrm{id}_{\Sigma_k}$$

is the identity map on Σ_k.

The success of the composition tree method depends heavily on finding effective epimorphisms onto smaller groups. This is where the geometric approach proves its value, since many of these epimorphisms are based on geometric structures preserved by the group. Some black box algorithms are also used, to identify (or name) simple groups and solve some constructive recognition and/or membership problems.

In addition to computing the composition tree of a matrix group, the LMG suite of functions also allows calculation of Sylow subgroups, derived subgroups and Fitting subgroups, amongst other things, as well as tests for nipotency and solubility.

6.4 New frontiers

Despite the significant advances in techniques for computing with matrix groups over the past 15 years, there are still important open questions in the area. We begin by outlining some of those which continue the aims of MGRP, before discussing some recent advances in computation in the Monster simple group.

Continuing the MGRP. Firstly, there is a demand for algorithms for determining the Aschbacher class of a matrix group with better theoretical and practical performance. Current algorithms have polynomial time complexity subject to the satisfaction of certain conditions, which are not always guaranteed. These algorithms mostly work well in practice, but can fail on some types of groups. Of particular focus are those recognising membership in classes C_4, C_6 and C_7.

The calculation of normalisers inside matrix groups also continues to be a challenging problem, as it is for permutation

groups.

There is also the matter of algorithms for matrix groups over infinite fields. As mentioned at the start of this section, decidability becomes a problem here and, moreover, these groups need not be finitely generated. The first algorithms in this area were developed in [4] and [7] in the 1990s, largely dealing with matrix groups over $\mathbb{Q}$. There have been significant innovations over the past 25 years, however, many fundamental questions still remain unanswered, such as subgroup membership testing. For more details we refer the reader to [27] for a survey of the current landscape in the field.

Computing in the Monster. The Monster $\mathbb{M}$ is the largest of the sporadic simple groups, with order approximately 8×10^{53}. Computing with the Monster is infeasible using standard methods, not necessarily because the Monster is so big (we can easily compute with S_{50}, which is larger), but because of its lack of a "reasonably sized" representation as a permutation or matrix group. Indeed, the smallest faithful permutation representation of $\mathbb{M}$ has degree around 10^{20}, while the smallest nontrivial matrix representation in any characteristic has dimension 196882. Therefore, even storing a set of generators for $\mathbb{M}$ in one of these formats takes a significant amount of space.

Despite these difficulties, Linton et al. achieved the first computer construction of the Monster in 1998 (see [63]), which represented a major breakthrough. Their construction is based on the restriction of the 196882-dimensional representation of $\mathbb{M}$ over $\mathbb{F}_2$ to the normaliser of an element of order 3. This restriction admits a decomposition of the representation into several smaller ones, allowing for more compact storage and faster multiplication of elements compared to in the larger matrix representation. An alternative implementation was developed by Holmes and Wilson [49], which instead considers a representation over $\mathbb{F}_3$ restricted to an involution centraliser. This has the added benefit of following the original construction of

the Monster by Griess [42] and the later simplification of Conway [25], allowing one to leverage the underlying geometry involved in those theoretical constructions.

These implementations had a significant impact on our understanding of the Monster, perhaps most notably through the discovery of several previously unknown maximal subgroups and information about other simple groups involved in the Monster. This work was largely carried out by Holmes and Wilson, in part because their computer construction was (necessarily) not publicly available, preventing use and verification by other members of the community. We refer the reader to [97] for an overview of these developments.

By the mid-2010s, there were still four open possibilities for (socles of) new maximal subgroups; however, progress stalled with Wilson's retirement and Holmes' departure from mathematics.

In 2020, after a career largely outside academia, Martin Seysen began the development of `mmgroup`, an open-source package to work with the Monster written in Python and C [90]. Like Holmes and Wilson's construction, Seysen's implementation is based on theoretical work of Griess and Conway. In this package, elements of $\mathbb{M}$ may be multiplied in a matter of milliseconds, orders of magnitude faster than those reported by Holmes and Wilson in their construction. The key to this improvement is a highly efficient word reduction algorithm, which allows the word problem in $\mathbb{M}$ to be solved very quickly.

In the couple of years since its release, `mmgroup` has been used to answer some long-standing questions about the Monster. Dietrich, Lee and Popiel [30] used it to complete the classification of the maximal subgroups of $\mathbb{M}$, while Pisani and Popiel [84] determined conjugacy class fusions from four maximal subgroups to $\mathbb{M}$. In a forthcoming paper, Dietrich et al. also find generators for all of the maximal subgroups of $\mathbb{M}$ (see [29]).

There are several further applications of `mmgroup` that could

be explored in the future, from computing further data about elements and subgroups of the Monster, to uses in the field of axial algebras. Integration of the functionality of `mmgroup` into GAP or Magma would also be desirable.

Acknowledgements

The author would like to thank the organisers of GABY2024 for their invitation to deliver the mini-course on which this survey article is based, as well as their hospitality and support during the summer school. She is also grateful to Nicola Grittini for his careful editing of the article. The author also acknowledges the support of an Australian Research Council Discovery Early Career Researcher Award (project number DE230100579).

REFERENCES

[1] H. Bäärnhielm – D. Holt – C.R. Leedham-Green – E.A. O'Brien: "A practical model for computation with matrix groups", *J. Symbolic Comput.* 68 (2015), 27–60.

[2] L. Babai: "Graph isomorphism in quasipolynomial time", in STOC'16: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, *Association for Computing Machinery*, New York, USA (2016), 684–697.

[3] L. Babai – R. Beals: "A polynomial-time theory of black box groups. I", *Cambridge University Press*, Cambridge (1999), 30–64.

[4] L. Babai – R. Beals – D. Rockmore: "Deciding finiteness of matrix groups in deterministic polynomial time", in Proceedings of the 1993 international symposium on Symbolic and algebraic computation, *Association for Computing Machinery*, New York, USA (1993), 117–126.

[5] L. Babai – R. Beals – Á. Seress: "Polynomial-time theory of matrix groups", in STOC'09: Proceedings of the forty-first annual ACM symposium on Theory of computing, *Association for Computing Machinery*, New York, USA (2009), 55–64.

[6] L. Babai – E. Szemeredi: "On the complexity of matrix group problems I", in 25th Annual Symposium on Foundations of Computer Science, *IEEE Computer Society Press* (1984), 229–240.

[7] R. Beals: "Algorithms for matrix groups and the Tits alternative", *J. Comput. Syst. Sci.* 58 (1999), 260–279.

[8] R. Beals – C.R. Leedham-Green – A.C. Niemeyer – C.E. Praeger – 'A. Seress: "Permutations with restricted cycle structure and an algorithmic application", *Combin. Probab. Comput.* 11 (2002), 447–464.

[9] K.D. Blaha: "Finding a minimum base for permutation groups is NP-hard", *Congr. Numer.* 58 (1987), 141–150.

[10] W.W. Boone: "The word problem", *Ann. of Math.* (2) 70 (1959), 207–265.

[11] W. Bosma – J. Cannon – C. Playoust: "The Magma algebra system. I. The user language", *J. Symbolic Comput.* 24 (1997), 235–265.

[12] S. Bratus – I. Pak: "Fast constructive recognition of a black box group isomorphic to S_n or A_n using Goldbach's conjecture", *J. Symbolic Comput.* 29 (2000), 33–57.

[13] M.R. Bridson – D.M. Evans – M.W. Liebeck – D. Segal: "Algorithms determining finite simple images of finitely presented groups", *Invent. Math.* 218 (2019), 623–648.

[14] M.R. Bridson – H. Wilton: "The triviality problem for profinite completions", *Invent. Math.* 202 (2015), 839–874.

[15] J.L. Britton: "The word problem for groups", *Proc. London Math. Soc.* (3) (1958), 493–506.

[16] T.C. Burness – R.M. Guralnick – J. Saxl: "On base sizes for symmetric groups", *Bull. Lond. Math. Soc.* 43 (2011), 386–391.

[17] T.C. Burness – R.M. Guralnick – J. Saxl: "Base sizes for $\mathcal{S}$-actions of finite classical groups", *Israel J. Math.* 199 (2014), 711–756.

[18] G. Butler: "Computational Approaches to Certain Problems in the Theory of Finite Groups", PhD thesis, *University of Sydney*, Sydney (1979).

[19] J.J. Cannon: "Construction of defining relators for finite groups", *Discrete Math.* 5 (1973), 105–129.

[20] J.J. Cannon – L.A. Dimino – G. Havas – J.M. Watson: "Implementation and analysis of the Todd–Coxeter algorithm", *Math. Comp.* 27 (1973), 463–490.

[21] J.J. Cannon – D.F. Holt – W.R. Unger: "The use of permutation representations in structural computations in large finite matrix groups", *J. Symbolic Comput.* 95 (2019), 26–38.

[22] F. Celler – C.R. Leedham-Green – S.H. Murray – A.C. Niemeyer – E.A. O'Brien: "Generating random elements of a finite group", *Comm. Algebra* 23 (1995), 4931–4948.

[23] M.S. Chang – C. Jefferson – C.M. Roney-Dougal: "Computing normalisers of intransitive groups", *J. Algebra* 605 (2022), 429–458.

[24] M.S. Chang – C.M. Roney-Dougal: "Primitive normalisers in quasipolynomial time", *Arch. Math.* (*Basel*) 118 (2022), 19–25.

[25] J.H. Conway: "A simple construction for the Fischer–Griess monster group", *Invent. Math.* 79 (1985), 513–540.

[26] M. Dehn: "Über unendliche diskontinuierliche Gruppen", *Math. Ann.* 71 (1911), 116–144.

[27] A.S. Detinko – D.L. Flannery: "Linear groups and computation", *Expo. Math.* 37 (2019), 454–484.

[28] H. Dietrich – A. Hulpke: "Universal covers of finite groups", *J. Algebra* 569 (2021), 681–712.

[29] H. Dietrich – M. Lee – A. Pisani – T. Popiel: "Explicit generators for maximal subgroups of the Monster" (2024); *arXiv*:2411.12230.

[30] H. Dietrich – M. Lee – T. Popiel: "The maximal subgroups of the Monster" (2023); *arXiv*:2304.14646.

[31] H. Dietrich – C.R. Leedham-Green – E.A. O'Brien: "Effective black-box constructive recognition of classical groups", *J. Algebra* 421 (2015), 460–492.

[32] A. Dietze – M. Schaps: "Determining subgroups of a given finite index in a finitely presented group", *Can. J. Math.* 26 (1974), 769–782.

[33] H. Duyan – Z. Halasi – A. Maróti: "A proof of Pyber's base size conjecture", *Adv. Math.* 331 (2018), 720–747.

[34] B. Eick – M. Horn: "Computing polycyclic quotients of finitely (L-)presented groups via Groebner bases", in Mathematical Software, *Springer*, Berlin (2010), 50–53.

[35] B. Eick – D. Kahrobaei: "Polycyclic groups: a new platform for cryptology?" (2004); *arXiv*:0411077

[36] J.B. Fawcett: "Bases of twisted wreath products", *J. Algebra* 607 (2022), 247–271.

[37] J.B. Fawcett: "The base size of a primitive diagonal group", *J. Algebra* 375 (2013), 302–321.

[38] J.B. Fawcett – J. Müller – E.A. O'Brien – R.A. Wilson: "Regular orbits of sporadic simple groups", *J. Algebra* 522 (2019), 61–79.

[39] H. Felsch: "Programmierung der Restklassenabzählung einer Gruppe nach Untergruppen", *Numer. Math.* 3 (1961), 250–256.

[40] The GAP Group: "GAP — Groups, Algorithms, and Programming", v4.13.1 (2024).

[41] V. Gebhardt: "Efficient collection in infinite polycyclic groups", *J. Symbolic Comput.* 34 (2002), 213–228.

[42] R.L. Griess Jr.: "The friendly giant", *Invent. Math.* 69 (1982), 1–102.

[43] J. Gryak – D. Kahrobaei: "The status of polycyclic group-based cryptography: a survey and open problems", *Groups Complex. Cryptol.* 8 (2016), 171–186.

[44] M.J.T. Guy: "Coset Enumeration", Lecture delivered at the Conference on Computational Problems in Abstract Algebra, Oxford, August 29 – September 2 (1967).

[45] Z. Halasi – M.W. Liebeck – A. Maróti: "Base sizes of primitive groups: bounds with explicit constants", *J. Algebra* 521 (2019), 16–43.

[46] G. Havas – M.F. Newman: "Application of Computers to Questions Like Those of Burnside", *Springer*, Berlin (1980).

[47] G. Havas – T. Nicholson: "Collection", in Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation, *Association for Computing Machinery*, New York, USA (1976), 9–14.

[48] G. Hiss – M. Neunhöffer – F. Noeske: "The 2-modular characters of the Fischer group Fi_{23}", *J. Algebra* 300 (2006), 555–570.

[49] P.E. Holmes – R.A. Wilson: "A new computer construction of the Monster using 2-local subgroups", *J. London Math. Soc.* (2) 67 (2003), 349–364.

[50] D.F. Holt – B. Eick – E.A. O'Brien: "Handbook of Computational Group Theory", *Chapman & Hall/CRC*, Boca Raton, FL (2005).

[51] D.F. Holt – W. Plesken: "Perfect Groups", *Oxford University Press*, New York, USA (1989).

[52] D.F. Holt – S. Rees: "Testing modules for irreducibility", *J. Aust. Math. Soc.* 57 (1994), 1–16.

[53] G. Ivanyos – K. Lux: "Treating the exceptional cases of the MeatAxe", *Experiment. Math.* 9 (2000), 373–381.

[54] S. Jambor: "An L3-U3-quotient algorithm for finitely presented groups", PhD thesis, *Aachen University*, Aachen (2012).

[55] S. Jambor: "An L_2-quotient algorithm for finitely presented groups on arbitrarily many generators", *J. Algebra* 423 (2015), 1109–1142.

[56] W.M. Kantor – K. Magaard: "Black box exceptional groups of Lie type II", *J. Algebra* 421 (2015), 524–540.

[57] M. Lee: "Regular orbits of quasisimple linear groups I", *J. Algebra* 586 (2021), 1122–1194.

[58] M. Lee: "Regular orbits of quasisimple linear groups II", *J. Algebra* 586 (2021), 643–717.

[59] J. Leech: "Coset enumeration on digital computers", *Proc. Cambridge Philos. Soc.* 59 (1963), 257–267.

[60] C.R. Leedham-Green: "A soluble group algorithm", in: M.D. Atkinson, *Academic Press*, London (1984), 85–101.

[61] C.R. Leedham-Green: "The computational matrix group project", in: Groups and Computation III, *de Gruyter*, Berlin (2001).

[62] C.R. Leedham-Green – L.H. Soicher: "Collection from the left and other strategies", *J. Symbolic Comput.* 9 (1990), 665–675.

[63] S. Linton – R. Parker – P. Walsh – R.A. Wilson: "Computer construction of the Monster", *J. Group Theory* 1 (1998), 307–337.

[64] E.H. Lo: "A polycyclic quotient algorithm", *J. Symbolic Comput.* 25 (1998), 61–97.

[65] F. Lübeck – M. Neunhöffer: "Enumerating large orbits and direct condensation", *Experiment. Math.* 10 (2001), 197–205.

[66] E.M. Luks: "Permutation groups and polynomial-time computation", in: Groups and Computation, *American Mathematical Society*, Providence, RI (1993).

[67] E.M. Luks – T. Miyazaki: "Polynomial-time normalizers", *Discrete Math. Theor. Comput. Sci.* 13 (2011), 61–96.

[68] I.D. Macdonald: "A computer application to finite p-groups'', *J. Aust. Math. Soc.* 17 (1974), 102–112.

[69] J. Mueller – M. Neunhöffer – F. Noeske – M. Horn: `orb`, "Methods to enumerate orbits", v.4.9.1 (Aug. 2024), GAP package;
url:`https://gap-packages.github.io/orb`.

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

[71] J. Neubüser: "Bestimmung der Untergruppenverbände endlicher p-Gruppen auf einer programmgesteuerten elektronischen Dualmaschine", *Numer. Math.* 3 (1961), 271–278.

[72] P.M. Neumann – C.E. Praeger: "A recognition algorithm for special linear groups", *Proc. London Math. Soc.* (3) 65 (1992), 555–603.

[73] M. Neunhöffer – Á. Seress: `recog`, "A package for constructive recognition of permutation and matrix groups", v.1.4.2 (Sep. 2022) GAP package; url:`https://github.com/gap-packages/recog`.

[74] M. Neunhöffer – Á. Seress: "A data structure for a uniform approach to computations with finite groups", in: ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation, *Association for Computing Machinery*, New York, USA (2006), 254–261.

[75] M.F. Newman – E.A. O'Brien: "Application of computers to questions like those of Burnside II", *Int. J. Algebra Comput.* 6 (1996), 593–605.

[76] W. Nickel: "Computing nilpotent quotients of finitely presented groups", in: Geometric and Computational Perspectives on Infinite Groups, *American Mathematical Society*, Providence, RI (1996), 175–191.

[77] A. Niemeyer: "A finite soluble quotient algorithm", *J. Symb. Comput.* 18 (1994), 541–561.

[78] A.C. Niemeyer – C.E. Praeger: "A recognition algorithm for classical groups over finite fields", *Proc. London Math. Soc.* (3) 77 (1998), 117–169.

[79] A.C. Niemeyer – C.E. Praeger: "A recognition algorithm for non-generic classical groups over finite fields", *J. Aust. Math. Soc.* 67 (1999), 223–253.

[80] P.S. Novikov: "On algorithmic unsolvability of the problem of identity", *Doklady Akad. Nauk SSSR* (*N.S.*) 85 (1952), 709–712.

[81] E.A. O'Brien: "Towards effective algorithms for linear groups", in: Finite Geometries, Groups, and Computation, *de Gruyter*, Berlin (2006), 163–190.

[82] E.A. O'Brien: "Algorithms for matrix groups", in: Groups St Andrews 2009, *Cambridge University Press*, Cambridge (2011), 297–323.

[83] R.A Parker: "The computer calculation of modular characters (the meat-axe)", in: Computational Group Theory, *Academic Press*, London (1984), 267–274.

[84] A. Pisani – T. Popiel: "Conjugacy class fusion from four maximal subgroups of the Monster", *J. Comput. Algebra*, 11 (2024), 5 pp.

[85] W. Plesken: "Towards a soluble quotient algorithm", *J. Symb. Comput.* 4 (1987), 111–122.

[86] W. Plesken – A. Fabiańska: "An L_2-quotient algorithm for finitely presented groups", *J. Algebra* 322 (2009), 914–935.

[87] L. Pyber: "Asymptotic results for permutation groups", in: Groups and computation, *American Mathematical Society*, Providence, RI (1993), 197–219.

[88] C.M. Roney-Dougal – S. Siccha: "Normalisers of primitive permutation groups in quasipolynomial time", *Bull. Lond. Math. Soc.* 52 (2020), 358–366.

[89] Á. Seress: "Permutation Group Algorithms", *Cambridge University Press*, Cambridge (2003).

[90] M. Seysen: "A fast implementation of the Monster group: The Monster has been tamed", *J. Comput. Algebra* 9 (2024).

[91] C.C. Sims: "Computational methods in the study of permutation groups", in: Computational Problems in Abstract Algebra, *Pergamon*, Oxford (1970), 169–183.

[92] C.C. Sims: "Some algorithms based on coset enumeration", unpublished notes, *Rutgers University*, New Brunswick, New Jersey, USA (1974).

[93] C.C. Sims: "Computation with Finitely Presented Groups", *Cambridge University Press*, Cambridge (1994).

[94] J.A. Todd – H.S.M. Coxeter: "A practical method for enumerating cosets of a finite abstract group", *Proc. Edinburgh Math. Soc.* 5 (1936), 26–34.

[95] H.F. Trotter: "A machine program for coset enumeration", *Canad. Math. Bull.* 7 (1964), 357–368.

[96] D. Wiebking: "Normalizers and permutational isomorphisms in simply-exponential time", in: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, *Association for Computing Machinery*, Philadelphia, PA (2020), 230–238.

[97] R.A. Wilson: "Maximal subgroups of sporadic groups", in: Finite simple groups: thirty years of the Atlas and beyond, *American Mathematical Society*, Providence, RI (2017), 57–72.

[98] R.A. Wilson: "Standard generators for sporadic simple groups", *J. Algebra* 184 (1996), 505–515.

Melissa Lee
School of Mathematics
Monash University
Clayton VIC 3800 (Australia)
e-mail: melissa.lee@monash.edu