Computing in Matrix Groups, and Constructive Recognition

Derek Holt

University of Warwick

LMS/EPSRC Short Instructional Course
Computational Group Theory
St Andrews
29th July – 2nd August 2013

Contents

Section 1

Introduction

Some notation

$o(g)$	The order of the group element g.
$\mathrm{Sym}(n)$	The symmetric group of degree n.
$\mathbf{S}_n$	A group isomorphic to $\mathrm{Sym}(n)$.
$\mathrm{Alt}(n)$	The alternating group of degree n.
$\mathbf{A}_n$	A group isomorphic to $\mathrm{Alt}(n)$.
$\mathrm{GL}_n(K)$	The group of all $n \times n$ matrices over the field K.
$\mathbb{F}_q$	The finite field of order q.
$\mathrm{GL}_n(q)$	$= \mathrm{GL}_n(\mathbb{F}_q)$.

As in GAP and Magma, matrices $g \in \mathrm{GL}_n(K)$ act on the right of row vectors $v \in K^n$, giving $vg \in K^n$.

Introduction

In this series of talks, we restrict ourselves to the study of computing in **finite** matrix groups.

This is mainly because there has been relatively little work to date on the development of algorithms for infinite matrix groups, although the time may be ripe to start thinking more about this topic.

In general, any finitely generated group $G \leq \mathrm{GL}_n(K)$ for a field K is either virtually solvable or it has a nonabelian free subgroup.
(This is known as the **Tits Alternative**.)

A recent algorithm of **Detinko, Flannery and O'Brien** [13] allows us to distinguish between these two cases.

Some progress has been made (by the same three authors) in carrying out effective computations in infinite virtually solvable groups, particularly in the virtually nilpotent case.

Restricting to finite fields

We shall also be restricting ourselves to matrix groups defined over **finite fields**.

We can justify this as follows.

In another paper by **Detinko, Flannery and O'Brien** [12], an effective algorithm is described to decide finiteness of a finitely generated group $G \leq \mathrm{GL}_n(K)$ where K is any infinite field that allows exact computation.

Let R be the subring of K generated by the entries in the generators of G.

When G is finite, the algorithm provides an explicit and effective monomorphism $\phi : G \to GL_n(R/\rho)$, where ρ is a maximal ideal of R, and R/ρ is a finite field.

Example

The quasisimple sporadic group $6.\mathrm{Suz}$ has a complex representation of degree 12, which can be written over the field $\mathbb{Q}(w)$, where w is a cube root of 1.

Starting from the image $G < \mathrm{GL}_{12}(\mathbb{Q}(w))$ of this representation, the Magma implementation proved finiteness of this group and found an isomorphic image of G in $\mathrm{GL}_{12}(25)$ in less than 0.1 seconds.

So we may assume from now on that $G \leq \mathrm{GL}_n(\mathbb{F}_q) = \mathrm{GL}_n(q)$ is defined over the **finite field $\mathbb{F}_q$ of order q**.

Of course, we may wish to lift the results of our calculations back to $\mathrm{GL}_n(K)$, for which purpose we need to compute inverse images under ϕ. This can be accomplished after solving the **rewriting problem** in the image, which will be discussed later.

Some history

Efficient algorithms for computing in finite permutations groups, based on the use of **base and strong generating set (BSGS)** methods date back to the 1970s.

As we shall see later, BSGS methods can also be used for some finite matrix groups, but to a far lesser extent.

It was observed by in a talk in the 1970s by **Charles Sims** that

$$|\mathrm{Sym}(100)| \sim 9.33 \times 10^{157} \quad \text{and} \quad |\mathrm{GL}_{15}(5)| \sim 1.41 \times 10^{157},$$

but it was possible at that time to compute very effectively in subgroups of Sym(100) and hardly at all in subgroups of $\mathrm{GL}_{15}(5)$.

It was allegedly widely believed for many years that there was nothing much to be done about this!

The Matrix Group Recognition Project

In the early 1990s, the

Matrix Groups Recognition Project (MGRP)

was officially instigated, with a view to remedying this unsatisfactory situation.

This project has been responsible for a huge number of published research papers, research grants, etc, covering both theoretical and practical aspects of computing in matrix groups, and of course lots of computer code.

It has even led to new directions in purely theoretical aspects of research in group theory, such as the statistical study of the proportions of elements in a group with various properties.

After more than 20 years we can report at least partial success. It remains much easier to compute in $\mathrm{Sym}(100)$ than in $\mathrm{GL}_{15}(5)$, but we can now at least solve basic problems in large finite matrix groups, such as identifying their composition factors.

Some fundamental difficulties

Two stumbling blocks affecting effective computation in matrix groups over $\mathbb{F}_q$ were recognized in the early days of CGT.

- We cannot even compute the order of an arbitrary element of $\mathrm{GL}_n(q)$.
- For some abelian groups $G = \langle x, y \rangle < \mathrm{GL}_n(q)$ of large prime exponent r, we cannot tell whether x is a power of y, so we do not know whether $|G| = r$ or r^2.

As we shall see shortly, the first problem can be solved effectively provided that we can factorize integers of the form $q^d - 1$ for $d \leq q$. A lot of effort has been put into computing and storing such factorizations. See [11]. Furthermore, it is not usually essential to know $o(g)$ exactly, and "a product of large primes dividing $q^d - 1$" is sufficient for most applications.

The second problem is an instance of the **discrete log** problem. Applications in cryptography have led to a huge amount of research on it, and it does not currently seem to be a bottleneck in applications.

C_r or $C_r \times C_r$?

Let $\alpha \in \mathrm{GL}_n(q)$ have order r for a large primitive prime divisor of $q^n - 1$.
Let $x, y \in \mathrm{GL}_{2n}(q)$ be defined by

$$x = \begin{pmatrix} \alpha^i & 0 \\ 0 & \alpha^j \end{pmatrix}, \quad y = \begin{pmatrix} \alpha^k & 0 \\ 0 & \alpha^l \end{pmatrix},$$

with $1 \leq i, j, k, l < r$.

Then $G = \langle x, y \rangle \cong C_r$ if $i/j \equiv k/l \bmod r$, and otherwise $G \cong C_r \times C_r$.

To decide this question, we need to find i/j and k/l, which require discrete log calculations in $\mathbb{F}_{q^n}$.

With $q = 7$, this calculation is fast with $n = 35$, $r = 77\,192\,844\,961$, but slow with $n = 40$, $r = 810\,221\,830\,361$.

Some basic algorithms

Powers of group elements. For a group G and $g \in G$, we can calculate a positive power g^n of g using $O(\log n)$ multiplications in G, by first computing $h := g^{\lfloor n/2 \rfloor}$ recursively and then $g^n = h^2$ or $h^2 g$.

In practice, we first write n in binary and then calculate g^n as the product of the appropriate g^{2^i}. For example, $g^{38} = g^{32} g^4 g^2$, which can be done with 7 group multiplications.

Orders of group elements with given multiplicative bound. If $g \in G$ and we know that the order $o(g)$ of g divides the factorized integer $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, then we can calculate $o(g)$ as follows.

1. if $k = 1$ then compute g^{p^i} for $i = 1, 2, \ldots$ until $g^{p^i} = 1$.
2. Otherwise write $n = n_1 n_2$ with n_1, n_2 coprime and having roughly equal factorization lengths, and calculate $|g|$ recursively as $o(g^{n_1})\, o(g^{n_2})$ (which divide n_2 and n_1 respectively).

Calculating the order of a matrix

Let $g \in \mathrm{GL}_n(q)$ with q a power of p. The following method due to **Celler and Leedham-Green** [6] can be used to calculate $o(g)$ provided that factorizations of the numbers $q^d - 1$ for $1 \leq d \leq n$ are available.

Algorithm: MatrixOrder

Input: $g \in \mathrm{GL}_n(q)$.

1. Calculate and factorize the minimal polynomial of g:
$$\mu(g) = f_1(x)^{\alpha_1} f_2(x)^{\alpha_2} \cdots f_k(x)^{\alpha_k}.$$
2. The p-part of $o(g)$ is p^β with $\beta = \lceil \log_p(\max \alpha_i) \rceil$.
3. Let $\{d_1, \ldots, d_j\}$ be the set of degrees of the polynomials $f_i(x)$. Then the p'-part of $o(g)$ is a factor of $\prod_{i=1}^{j}(q^{d_i} - 1)$.

Section 2

Homomorphisms, rewriting and Straight Line Programs

Homomorphisms

Definition

We call a homomorphism ϕ defined on a permutation or a matrix group G **directly computable** if there is an efficient method of evaluating $\phi(g)$ directly from g, for all $g \in G$.

Setting up the algorithm to evaluate ϕ may involve some preprocessing involving the group G, but after that, $\phi(g)$ must be computable from g alone, without further reference to the G.

Example

If G is an intransitive or imprimitive permutation group, then its induced action on one of its orbits, or its induced action on the blocks of imprimitivity is a directly computable homomorphism.

Example

If $G < \mathrm{GL}_n(q)$ is a reducible matrix group that fixes a subspace W of $V = \mathbb{F}_q^n$ then, provided we know a basis for W, the induced actions of G on V and on W/V are directly computable homomorphisms.

Example

If $G < \mathrm{GL}_n(q)$ is an imprimitive matrix group that preserves a decomposition $V_1 \oplus \cdots \oplus V_k$ of $V = \mathbb{F}_q^n$ then, provided we know bases for each V_i, the induced permutation action of G on $\{V_1, V_2, \ldots, V_n\}$ is directly computable.

Rewriting and Straight Line Programs

Some homomorphisms are not directly computable, and are defined by the specification of $\phi(x)$ for x in a generating set X of G.

To evaluate such homomorphisms on arbitrary $g \in G$, we require algorithms to write arbitrary elements of G as words over $X \cup X^{-1}$. This is known as the **rewriting problem**.

In practice, it may not be possible to do this explicitly, because the words would be to long.

This difficulty is overcome by the use of **Straight Line Programs** (**SLP**s).

Definition

Let X be a finite set. Then an **SLP** over X is a sequence $f_1, \ldots, f_k$ of formal expressions such that, for each i, either

- $f_i = x$ or $f_i = x^{-1}$ with $x \in X$; or
- $f_i = f_j^{\pm 1} f_k^{\pm 1}$ with $j, k < i$.

Example

$X = \{x_1, x_2\}$:

$$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1x_2, \quad f_4 = f_3^2, \quad f_5 = f_2^2, \quad f_6 = f_4^{-1}f_2, \quad f_7 = f_5f_6.$$

Definition

If X is a subset of a group G and σ is an SLP over X, then **Eval(σ)** is the sequence of elements of G defined by σ.

Example

With σ as above and F the free group on X, Eval(σ) is the sequence:

$$x_1, \quad x_2, \quad x_1x_2, \quad (x_1x_2)^2, \quad x_2^2, \quad (x_2^{-1}x_1^{-1})^{-1}x_2, \quad x_2x_1^{-1}x_2^{-1}x_1^{-1}x_2.$$

Definition

For a group $G = \langle X \rangle$ and $g \in G$, **writing g as an SLP over X** means finding an SLP $\sigma = (f_1, \ldots, f_k)$ over X with $\text{Eval}(\sigma)_k = g$.

Definition

If σ is an SLP over X and $\phi : X \to Y$ is a map, then $\phi(\sigma)$ is the SLP over Y defined by replacing each term $x_i^{\pm 1}$ in σ by $\phi(x_i)^{\pm 1}$.

So, if $G = \langle X \rangle$, σ is an SLP over X, and $\phi : G \to H$ is a homomorphism, then $\phi(\mathrm{Eval}(\sigma)) = \mathrm{Eval}(\phi(\sigma))$.

So, if we can write $g \in G$ as an SLP over X, then we can write $\phi(g)$ as an SLP over $\phi(X)$, and evaluate it if required. We can also use this mechanism to evaluate inverse images of elements under homomorphisms.

For many types of subgroups $G \leq \mathrm{Sym}(n)$ or $G \leq \mathrm{GL}_n(q)$, we can find generating sets Y for which there are efficient methods for writing elements of G as words (or SLPs) over Y. For example, Y may be a *strong generating set* w.r.t to a *base* of G.

So, if we are given $G = \langle X \rangle$, and we can find the elements of Y as SLPs over X, then we can express all elements $g \in G$ as SLPs over X.

Example

We have $\mathrm{Alt}(6) = \langle X \rangle$ with $X = \{x_1, x_2\}$, $x_1 = (1,2,3)$, $x_2 = (2,3,4,5,6)$.

Then, for the SLP σ as above, $\mathrm{Eval}(\sigma))$ is the sequence $g_1, \ldots, g_7 =$

$$x_1, \ , x_2, \ (1,3)(2,4,5,6), \ (2,5)(4,6), \ (2,4,6,3,5), \ (2,6,5,3,4), \ (4,5,6).$$

In fact $\{g_1, g_2, g_4, g_7\}$ is a strong generating set for $\mathrm{Alt}(6)$ w.r.t. the base $1,3,2,4$, and we can write elements of $\mathrm{Alt}(6)$ as words in the strong generators.

For example, let $g = (1,2,5,4)(3,6)$.

Then $g = g_7 g_2^{-2} g_1$, so $(f_1, \ldots, f_{10})$ is an SLP for g over X, with f_i as before for $1 \leq i \leq 7$, and

$$f_8 = f_7 f_2^{-1}, \ \ f_9 = f_2^{-1} f_1, \ f_{10} = f_8 f_9.$$

Section 3

Probabilistic algorithms

Black box groups

Definition

A **black box group** is a group $G = \langle X \rangle$ whose elements are represented by binary strings of fixed length N. It is not guaranteed that every such string represents a group element, and a given group element may be represented by more than one string. We are given strings that represent the generators X, and also a string representing the identity element of G.

Furthermore, we are given oracles that can perform the following operations in constant time.

1. Given two strings representing group elements g and h, return a string representing gh.
2. Given a string representing a group element g, return a string representing g^{-1}.
3. Given two strings representing group elements g and h, decide whether $g = h$.

Remark

In this definition, **black box** describes the method of representation of G rather than specifying a property of G as abstract group.
In practice, the strings used to represent group elements need not be binary, but they must have bounded length.

Example

Subgroups $G = \langle X \rangle$ of $\mathrm{Sym}(n)$ or $\mathrm{GL}_n(q)$ can be regarded as black box groups.

It is often useful to devise algorithms for black box groups, possibly provided with extra oracles, such as a fast oracle for computing orders of group elements, since this can extend their range of applicability.

Such an algorithm can be applied both to permutation and to matrix groups.

Choosing random elements of groups

A large number of algorithms in CGT involve choosing random elements from groups, and for the level of generality required, algorithms designed to work in black box groups are the most suitable.

A polynomial time algorithm for doing this was described by Babai in 1991, but it is too slow for practical purposes. (Polynomial of degree 10 pre-processing, and degree 5 for each random element.)

The **Product Replacement Algorithm** of **Charles Leedham Green** [7] provides a satisfactory compromise between true randomness and speed of execution.

It is very simple to describe and implement, and it has the advantage that SLPs over X of the random elements returned are easily calculated.

The Product Replacement Algorithm

Algorithm: Product Replacement

Input: A black box group $G = \langle X \rangle$ and parameters n and c.
Default: $n = 10$, $c = 50$.

1. Define list Y: Let Y be an ordered list with $|Y| \geq n$ containing the elements of X, with elements repeated if $|X| \leq n$.

   Basic Move: Choose random distinct i, j with $1 \leq i, j \leq n$ and replace $Y[i]$ by one of the four elements

   $$Y[j]Y[i], \quad Y[j]^{-1}Y[i], \quad Y[i]Y[j], \quad Y[i]Y[j]^{-1},$$

   chosen at random.

2. Initialization: Perform the basic move c times.

3. Generate a random element: Perform the basic move and then return $Y[i]$.

Probabilistic algorithms

A **probabilistic algorithm** is one that involves random choices; that is, calls to a random number generator. There are two types of these that arise frequently in CGT.

Definition

A **Monte Carlo** algorithm takes a real number $\epsilon \in (0, 1)$ as an additional input parameter. It always returns an answer, which may be incorrect, but the probability of it being incorrect on any input must be less than ϵ.

Definition

A **Las Vegas** algorithm also takes a real number $\epsilon \in (0, 1)$ as an additional input parameter. It does not always return an answer but, if it does, then the answer is guaranteed to be correct. The probability of it failing to give an answer on any input must be less than ϵ.

Example (A Monte-Carlo algorithm to test whether a given group $G = \langle X \rangle \leq \mathrm{Sym}(n)$ contains $\mathrm{Alt}(n)$.)

It was proved by **Jordan** in 1873 that a transitive subgroup of $\mathrm{Sym}(n)$ that contains an element that has a cycle of prime order p with $n/2 < p < n-2$ is equal to $\mathrm{Alt}(n)$ or $\mathrm{Sym}(n)$.

The proportion $d(n)$ of elements of $\mathrm{Alt}(n)$ or $\mathrm{Sym}(n)$ with this property can be estimated. (It is roughly $\log 2 / \log n$ for large n).

This justifies the following simple one-sided Monte-Carlo algorithm for testing whether $G = \langle X \rangle \leq \mathrm{Sym}(n)$ is equal $\mathrm{Alt}(n)$ or $\mathrm{Sym}(n)$.

1. Given ϵ and $G = \langle X \rangle \leq \mathrm{Sym}(n)$, test if G is transitive, and return FALSE if not.
2. Choose $-\log \epsilon / d(n)$ random elements of G. If any of these contain a p-cycle for a prime p with $n/2 < p < n-2$ then return TRUE. Otherwise return FALSE

Note that the answer TRUE is definitely correct, but the answer FALSE has a probability of at most ϵ of being wrong.

Section 4

Base and strong generating set methods in matrix groups

Base and strong generating set methods in matrix groups

The important idea of a **base and strong generating set (BSGS)** in a permutation group has been covered in the lectures on permutation groups. It can be generalized to matrix groups as follows (**Butler** [5]).

As usual, let $G \leq \mathrm{GL}_n(q)$ act on $V = \mathbb{F}_q^n$.
Let α be either a vector $v \in V$ or a subspace $W \leq V$.
In either case, we can define the orbit and the stabilizer:

$$\alpha^G = \{\alpha g \mid g \in G\}, \qquad G_\alpha = \{g \in G \mid \alpha g = \alpha\}$$

of G on α.

If $(\alpha_1, \ldots, \alpha_k)$ is a sequence in which each such α_i is a vector in V or a subspace of V, then we define the stabilizer $G_{\alpha_1,\ldots,\alpha_k}$ of the sequence to be the intersection of the stabilizers G_{α_i} for $1 \leq i \leq k$.

We define the associated **stabilizer chain** as

$$G = G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(k)} \geq G^{(k+1)}$$

where $G^{(i+1)}$ is defined to be the stabilizer of the subsequence $(\alpha_1, \ldots, \alpha_i)$ for $1 \leq i \leq k$.

The sequence is called a **base** for G if its stabilizer is trivial; that is, if $G^{(k+1)} = 1$.

In that case, we have the associated sequence of **basic orbits** $\Delta^{(i)} = \alpha_i^{G^{(i)}}$, for $1 \leq i \leq k$.

A subset S of G is called a **strong generating set** with respect to this base, if $\langle S \cap G^{(i)} \rangle = G^{(i)}$ for $1 \leq i \leq k$.

Note that these definitions are the same as the corresponding definitions for permutation groups.

Most implemented algorithms for finding a base, use a combination of 1-dimensional subspaces $\langle v \rangle$ and vectors v as base points.

The vectors are only needed to handle scalar matrices in G, which stabilize all subspaces of V.

Given a BSGS for a matrix group, modulo some technicalities, more or less the same algorithms, including backtrack search methods, can be used for to carry out structural computations in the matrix group G. Their theoretical complexity and practical effectiveness is strongly dependent on the lengths $|\Delta^{(i)}|$ of the basic orbits

Although this is no longer the only approach available for computing in matrix groups, it remains our preferred method provided that we can find a base with moderately short basic orbits.

Finding a good base

The **Schreier-Sims** method (and its variants, such as **Todd-Coxeter Schreier-Sims**) can be used to find and verify a BSGS, but there are two serious problems that apply to matrix groups but not to permutation groups.

- There may not exist a base with short basic orbits. For example, if $G = \mathrm{GL}_n(q)$, and $\alpha = \langle v \rangle$ is a 1-dimensional subspace of V, then $|\alpha^G| = (q^n - 1)/(q - 1)$, and any base for G must include a basic orbit of at least this length.
- There may exist a base for G with reasonably short basic orbits, but it might be difficult to find.

There is nothing we can do about the first of these. We need to find other methods of computing in groups without moderately short basic orbits.

Looking for short basic orbits

For the second problem, we can try and devise improved methods for finding vectors v such that α^G is short, with $\alpha = \langle v \rangle$.

If $|\alpha^G|$ is small, then $|G_\alpha|$ is large, and v is an eigenvector of all elements in G_α.

Murray and O'Brien [21] used this idea to devise an improved search for good base points:

Choose a collection of random elements $g_i \in G$, and calculate their eigenvectors. If two or more g_i have a common eigenspace $\langle v \rangle$, then consider $\alpha = \langle v \rangle$ as a possible base point.

This approach resulted in significantly improved performance of BSGS methods in matrix groups.

For specific groups, it can be worthwhile devoting some effort to finding a good base, and we need not restrict ourselves to subspaces of dimension 1.

Example

The alternating groups $\mathbf{A}_n$ with $n \geq 5$ have a quasisimple double cover $2.\mathbf{A}_n$, which has centre Z of order 2 with $2.\mathbf{A}_n/Z \cong \mathbf{A}_n$. These group do not have even moderately low degree faithful permutation representations, but they do arise as matrix groups of moderate degree.

For example, $G = 2.\mathbf{A}_{15}$ has no faithful permutation representation of degree less than 10^7, but $G \leq \mathrm{GL}_{64}(3)$.

This group has a base (α_i) of lengths 10, with basic orbit lengths
210, 13, 12, 11, 10, 9, 8, 7, 360, 2
where the first 9 base points are subspaces of dimensions
32, 16, 16, 16, 8, 8, 8, 4, 1
and the final base point is a vector.

Using this base, we can compute effectively in G.

Section 5

Aschbacher's Theorem for matrix groups

We now move on to consider how to compute in matrix groups for which no suitable BSGS can be found.

We start with a few definitions.

A group G is **perfect** if it is equal to its own commutator subgroup $[G, G]$.

A group G is **quasisimple** if G is perfect and $G/Z(G)$ is a nonabelian simple group.

In that case $Z(G)$ is a quotient group of the Schur Multiplier of $G/Z(G)$ and G itself is a central quotient of the covering group of $G/Z(G)$.

For example, the covering group of $\mathrm{PSL}_n(q)$ is, except in one or two exceptional cases, $\mathrm{SL}_n(q)$.

A group G is **almost simple** if it has a nonabelian simple normal subgroup S with $C_G(S) = 1$.

In that case G is isomorphic to a subgroup of $\mathrm{Aut}\, S$.

Aschbacher's Theorem for matrix groups

The **O'Nan Scott Theorem**, which classifies finite permutation groups into types (intransitive, imprimitive, affine, almost simple, product type, etc.), is used in many of the more advanced algorithms for computing in permutation groups. It also plays a fundamental role in the classification of maximal subgroups of $\mathrm{Alt}(n)$ and $\mathrm{Sym}(n)$.

There is a corresponding result, due to **Michael Aschbacher** [1], for subgroups of $\mathrm{GL}_n(q)$. The aim was to provide a foundation for the study of the maximal subgroups of the simple classical groups, and this project was continued in the book by **Kleidman and Liebeck** [20].

The version of **Aschbacher's Theorem** that has been used extensively in modern algorithms for computing in subgroups of $\mathrm{GL}_n(q)$ is much less detailed than the full theorem, and had been proved earlier by **Robert Wilson**, who in turn attributes it in his book [25] to Dynkin. These proofs are related to Clifford's Theorem.

It is this less detailed version that we shall now state.

Statement of Aschbacher's Theorem

Theorem

Let $G \leq \mathrm{GL}_n(q)$ acting on $V := \mathbb{F}_q^n$. Then either

1. *G is of* **geometric type**, *and G satisfies (at least) one of the conditions* C1–C7 *listed below; or*
2. [Condition C8] *There exists $N \trianglelefteq G$ with N a classical group in its natural representation.*
3. [Condition C9/$\mathcal{S}$] *G^∞ acts absolutely irreducibly on V, $G/Z(G)$ is almost simple, G^∞ is not conjugate in $\mathrm{GL}_n(q)$ to a subgroup of $\mathrm{GL}_n(r)$ with $r < q$, and G^∞ is not equal to a classical group in its natural representation. (So G does not satisfy* C1, C3, C5, or C8.)

It is more usual to regard C8 as being of geometric type. The maximal subgroups satisfying one of C1–C8 are generic, and can be described uniformly in all dimensions. Those satisfying C9 can only be classified on a case-by-case basis.

C1 G acts reducibly on V. Then $G \leq p^{kl}.(\mathrm{GL}_k(q) \times \mathrm{GL}_l(q))$ with $k + l = n$.

C2 G acts imprimitively on V. Then $G \leq \mathrm{GL}_k(q) \wr \mathrm{Sym}(n/k)$ with k a proper divisor of n.

C3 G acts semilinearly on V. Then $G \leq \mathrm{GL}_k(q^{n/k}).C_{n/k}$, with k a proper divisor of n.

C4 G preserves an inhomogeneous tensor product decomposition of V. Then $G \leq \mathrm{GL}_k(q) \circ \mathrm{GL}_{d/k}(q)$ with $1 \neq k$ a proper divisor of n.

C5 G is conjugate in $\mathrm{GL}_n(q)$ to a subgroup of $\mathrm{GL}_n(r)Z$ for a proper divisor r of q, where $Z := Z(\mathrm{GL}_n(q))$.

C6 There exists $P \trianglelefteq G$ with P an extraspecial r-group or a 2-group of symplectic type. Then $G \leq r^{1+2k}.\mathrm{Sp}_{2k}(r)Z$ with $n = r^k$.

C7 G preserves a homogeneous tensor product decomposition of V. Then $G \leq \mathrm{GL}_k(q) \wr_\circ \mathrm{Sym}(r)$ with $n = k^r$.

We say that a subgroup of $\mathrm{GL}_n(q)$ has **Type** C1, etc, if it satisfies Condition C1.

The maximal subgroups of Types C1–C8 of all almost simple extensions of classical groups in dimensions greater than 12 are classified in great detail in the book by Kleidman and Liebeck [20].

For subgroups of geometric type (C1–C7), there is an associated directly computable homomorphism from G to a group H that is one of:

- a cyclic group of small order;
- a subgroup of $\mathrm{Sym}(d)$ with $d \leq n$; or
- a subgroup of $\mathrm{GL}_d(r)$ (or $\mathrm{PGL}_d(r)$) with either $d < n$ or $r < q$.

We mentioned these earlier for reducible and imprimitive subgroups (Types C1 and C2).

Aschbacher reductions

More precisely, in the seven case, we have:

Type C1: $H = \mathrm{GL}_k(q)$ or $\mathrm{GL}_l(q)$.

Type C2: $H = \mathrm{Sym}(k/l)$.

Type C3: Either $H = C_l$ with l dividing n/k, or $H = \mathrm{GL}_k(q^{n/k})$.

Type C4: $H = \mathrm{PGL}_k(q)$ for some proper divisor k of n.

Type C5: $H = \mathrm{PGL}_n(r)$.

Type C6: $H \leq \mathrm{Sp}_{2k}(r)$.

Type C7: $H = \mathrm{Sym}(r)$.

Algorithms to test for the conditions C1–C8

A variety of methods have been developed over the past 20 years to test whether $G \leq \mathrm{GL}_n(q)$ satisfies one of the conditions C1–C8. These are mostly probabilistic algorithms.

The paper by **Neumann and Praeger** [22] describes a Monte-Carlo algorithm for testing whether $\mathrm{SL}_n(q) \leq G$.

This result is generally regarded as being the first in the **MGRP**. It arose from a question posed by **Joachim Neubüser** in 1988, as to whether there is an analogous algorithm for matrix groups to the well-known Monte-Carlo algorithm for testing whether a permutation group of degree n contains $\mathrm{Alt}(n)$.

It was generalized by **Niemeyer and Praeger** in [24] to a Monte-Carlo algorithm to test whether G normalizes a classical group; that is, to a test for Condition **C8**.

Testing for C5

An efficient Las-Vegas algorithm to test for Condition **C5** (writeable mod scalars over proper subfield) is described in a paper by **Glasby, Leedham-Green and O'Brien** [14].

The idea is that G is conjugate to a subgroup of $\mathrm{GL}(n, r)$ for $\mathbb{F}_r < \mathbb{F}_q$ if and only if the characteristic polynomial $c(g)$ lies in $\mathbb{F}_r[x]$ for all $g \in G$.

If so, then to find a conjugating matrix, we first find $g \in G$ that has an eigenvalue of multiplicity 1 in $\mathbb{F}_r$ and construct a basis of $\mathbb{F}_r^n$ consisting of images of a corresponding eigenvector under elements of G.

The MeatAxe

Algorithms to test for **C1** (reducibility) pre-date **MGRP**. The original version of the **MeatAxe** Algorithm was due to **Richard Parker** in about 1982 (but **John Conway** is responsible for the name).

It was used to construct irreducible modules of (typically sporadic) simple groups over small fields as constituents of permutation modules and tensor products of known modules.

A slightly modified version that is practical for arbitrary finite fields is described by **Holt and Rees** in [17], and this provides a satisfactory test for Type C1.

These are all Las-Vegas algorithms. They work by choosing random elements of the matrix algebra generated by the generators of G, until an element is found, for which the characteristic polynomial has certain properties. This element can then be used to decide reducibility.

The proportion of all elements of the matrix algebra that have the required property can be estimated, and so we can compute a number N such that the probability of N random elements of the algebra containing such an element is at least the given number ϵ.

If we do not find such an element, then we can either report failure or (as is often done in implementations) just try again.

Of course, this assumes that we can choose random elements of the algebra!

There are a number of other **MeatAxe**-related algorithms:

- Testing irreducible $\mathbb{F}_q G$-modules for absolute irreducibility; (a partial test for **C3**).
- Testing two irreducible $\mathbb{F}_q G$-modules for isomorphism;
- Other module-theoretic calculations, such as finding the socle or Jacobson radical of an $\mathbb{F}_q G$-module.

Smash

The algorithm **Smash** described in [16] provides a fast method of showing that G satisfies one of the conditions **C2, C3, C4, C6**, or **C7**, provided that we are given nontrivial elements of the kernel of the associated directly computable homomorphism $\phi : G \to H$.

It makes use of the **MeatAxe** and related algorithms, and it corresponds roughly to parts of the proof of simple versions of Aschbacher's Theorem based on Clifford's Theorem.

Algorithm: Smash

Input: $G = \langle X \rangle \leq \mathrm{GL}_n(q)$ with G acting absolutely irreducibly on its natural $\mathbb{F}_q G$-module V, and $Y \subset G$.

Put $N := \langle Y^G \rangle$ and let V_N be its natural $\mathbb{F}_q N$-module.

1. Write $V_N \cong \oplus_{i=1}^{k} V_i$, for irreducible $\mathbb{F}_q N$-modules V_i.

2. If the V_i are not all isomorphic, then the homogeneous components in the direct sum decomposition form a system of blocks of imprimitivity for N and G satisfies **C2**.

   So assume that the V_i are all isomorphic.

3. If the V_i are not absolutely irreducible, then G satisifies **C3**.

   So assume that the V_i are absolutely irreducible.

4. If $i > 1$ then G satisifies **C4**.

5. If $i = 1$ and N is abelian, then then G satisifies **C6**.

6. If $i = 1$ and N is non-abelian, then then G satisifies **C7**.

In each of the above cases, we can define the directly computable homomorphism ϕ associated with the Aschbacher type, and $N \leq \ker \phi$.

Thus the problem of finding Aschbacher reductions is reduced to finding nontrivial elements of the kernels of their associated directly computable homomorphisms.

Several papers have been published describing and analysing methods of doing this in the different cases.

These have not yet been proven to work effectively in all cases but, on the whole, they appear to perform satisfactorily on typical examples.

Of course, in some examples, G may satisfy one of the conditions, such as C2 or C4, with $\ker \phi$ trivial, in which case **Smash** has no hope of succeeding.

If $\ker \phi$ is trivial for all such decompositions, then in fact G is of type **C9** and can be considered as such.

Section 6

Recognition algorithms

Recognition algorithms

We now consider matrix groups of type **C8** or **C9**.

Definition

Let $\mathcal{X}$ be a set of subgroups of $\mathrm{Sym}(n)$ for some n, or of $\mathrm{GL}_n(q)$ for some n and q.
A **non-constructive recognition** algorithm for $\mathcal{X}$ is an algorithm (possibly Monte-Carlo or Las Vegas) which, given any subgroup G of $\mathrm{Sym}(n)$ or of $\mathrm{GL}_n(q)$, decides whether $G \in \mathcal{X}$.

For example, the one-sided Monte Carlo algorithm described earlier for deciding whether a given subgroup of $\mathrm{Sym}(n)$ contains $\mathrm{Alt}(n)$ is a non-constructive recognition algorithm for $\{\mathrm{Alt}(n), \mathrm{Sym}(n)\}$.

The Niemeyer-Praeger algorithm for testing for Condition C8 is also a one-sided Monte Carlo non-constructive recognition algorithm for the classical groups and their normalizers in their natural representations.

Standard copies

Definition

A **standard copy** of a group G is a fixed member of its isomorphism class.

Example

Fort the standard copy of the symmetric group $\mathbf{S}_n$, we might choose the group $\mathrm{Sym}(n)$ of all permutations of the set $\{1, \ldots, n\}$.

Example

For an abelian group, we might choose the standard PC-presentation of that group as its standard copy.

In general, we try to choose the most 'natural' instance of a group for its standard copy, although there may sometimes be more than one candidate.

Two types of standard copies are used in practice for groups that are close to being nonabelian simple.

(i) A subgroup of $\mathrm{Sym}(n)$ for some n.

Example

We would choose this for some of the sporadic simple groups, such as the Mathieu groups.

(ii) A subgroup of $\mathrm{GL}_n(q)$ for some n and q or, more generally, a quotient of a subgroup of $\mathrm{GL}_n(q)$ by a subgroup N, where N is usually a group of scalars.

Example

We would use this for the classical simple groups. For example, for the standard copy of $\mathrm{PSL}_n(q)$, we would choose the quotient group $\mathrm{SL}_n(q)/Z(\mathrm{SL}_n(q))$. Elements of the standard copy are represented by matrices in $\mathrm{SL}_n(q)$, which are regarded as being cosets of the scalar subgroup.

Standard generators and constructive recognition

Together with the standard copy of a group, we may also choose some specific generators of the standard copy, which we call the **standard generators**.

They should be chosen with a view to enable efficient algorithms for solving the rewriting problem for the group on those generators.

Example

We choose $(1, 2)$ and $(1, 2, 3, \ldots, n)$ as standard generators for $\mathrm{Sym}(n)$.

Definition

A **constructive recognition algorithm** for a group G constructs an isomorphism ϕ from G to the standard copy S of G, such that images of elements under ϕ and ϕ^{-1} can be effectively computed.

We assume, as usual, that the group G is defined by means of a generating set X. It may be defined either as a black box group, or as subgroup of some $\mathrm{Sym}(n)$ or of $\mathrm{GL}_n(q)$ (although, in the second case, we may still choose to regard it as a black box group).

If G is defined as a subgroup of $\mathrm{Sym}(n)$ or of $\mathrm{GL}_n(q)$, then a constructive recognition algorithm should also be able to solve the **constructive membership problem** for G:

Given $g \in \mathrm{Sym}(n)$ or $\mathrm{GL}_n(q)$, decide whether $g \in G$ and, if so, write g as an SLP over the given generators X of G.

A constructive recognition algorithm for G will typically not attempt to verify that the input group is really isomorphic to G, and will assume that this is the case.

In practice, this assumption may result from the output of a Monte Carlo algorithm, and so there may be a small probability that it is false. In that case, the output of the algorithm will be unpredictable.

The general strategy of constructive recognition algorithms is to start by using random methods to find elements X', defined as SLPs over the given generators X of G, that map under ϕ to the standard generators Y of S.

We then need a method of computing $\phi(g)$ for arbitrary $g \in G$ which, typically, will not use rewriting techniques. (It may depend, for example, on the order of g and of gx for $x \in X'$.)

If we have an algorithm for rewriting elements of S as SLPs σ over the standard generators Y, then we can write their images under ϕ^{-1} as SLPs $\phi^{-1}(\sigma)$ over X', and evaluate them as Eval$(\phi^{-1}(\sigma)$.

Since the elements X' were defined as SLPs over X, we can now write elements $g \in G$ as SLPs over X by writing $\phi(g)$ as an SLP σ over Y and then rewriting the SLP $\phi^{-1}(\sigma)$ over X' as an SLP over X.

Typically, if we attempt to evaluate $\phi(g)$ with $g \notin G$ then either the method will report failure, which will prove that $g \notin G$, or the rewriting algorithm for $\phi(g)$ will report failure, which will prove that $\phi(g) \notin S$ and hence $g \notin G$.

Constructive recognition of $\mathbf{S}_n$

As an example, we shall now briefly describe a constructive recognition algorithm for the symmetric groups $\mathbf{S}_n$ for black box groups with order oracle. There is a similar method for $\mathbf{A}_n$. This method is due to **Bratus and Pak** [4]. It works as described here for $n \geq 20$ and requires minor modifications for smaller n.
We use $(1, 2)$ and $(1, 2, 3, \ldots, n)$ as standard generators of $S = \mathrm{Sym}(n)$.

The method relies on a strong form of Goldbach's Conjecture: we need even numbers to be expressible in many ways as a sum of two primes. This certainly applies to all numbers within the practical range of the algorithm.

We start by looking for an element $g \in G$ of order pq for distinct odd primes p and q with $p + q = n$ or $p + q = n - 1$, and let g_1 and g_2 be powers of g of orders p and q, which will map onto a single p-cycle and a single q-cycle in S.

The proportion of such elements in $\mathbf{S}_n$ is reasonably high. In $\mathbf{S}_{20}$, $\mathbf{S}_{51}$, it is about 3% and 1.5%. It is conjectured to be $\Theta(\log\log n/(n \log n))$.

Example

With $n = 20$, we might find g with
$g \mapsto (1,2,3,4,5,6,7,8,9,10,11,12,13)(14,15,16,17,18,19,20) \in S$,
$g_1 = g^7 \mapsto (1,8,2,9,3,10,4,11,5,12,6,13,7)$,
$g_2 = g^{13} \mapsto (14,20,19,18,17,16,15)$.

Now we look for an element t mapping onto a transposition of S. We find t as power of a randomly chosen element h of order $2p_1q_1$ with p_1, q_1 distinct odd primes with $2 + p_1 + q_1 = n$ when n is even, or $6p_1q_1$ with p_1, q_1 distinct primes greater than 3 and $5 + p_1 + q_1 = n$ when n is odd.

Example

With $n = 20$, we might find
$h \mapsto (1,19,20,13,2,5,17)(3,6,8,12,15,18,16,14,9,4,10)(7,11) \in S$,
and then define $t := h^{77} \mapsto (7,11)$.

Next, we look for an element u that maps to an n-cycle of S.

When n is even, we choose $u = g_1 t^h g_2$, where t^h is a random conjugate of t that does not commute with g_1 or g_2 (and hence $t^h \mapsto (i, j)$ with i in the p-cycle g_1 and j in the q-cycle g_2).

When n is odd, we choose $u = g_1 t^h g_2 t^{h'}$ for two suitably chosen random conjugates of t.

Example

With $n = 20$, $t \mapsto (7, 11)$, $g_1 \mapsto (1, 8, 2, 9, 3, 10, 4, 11, 5, 12, 6, 13, 7)$, $g_2 \mapsto (14, 20, 19, 18, 17, 16, 15)$, after choosing one or two random conjugates of t, we found $t^h \mapsto (2, 15)$, which commutes with neither g_1 nor g_2. Then
$u = g_1 t^h g_2 \mapsto (1, 8, 14, 20, 19, 18, 17, 16, 15, 2, 9, 3, 10, 4, 11, 5, 12, 6, 13, 7)$.

Since $t^h \mapsto (i, j)$, where i and j are adjacent points in the n-cycle mapped to by u, we can now replace t by t^h and assume that

$$t \mapsto (1, 2), \quad u \mapsto (1, 2, 3, \ldots, n).$$

This achieves the first aim of finding elements of G that map onto the standard generators of $S = \mathrm{Sym}(n)$ under the isomorphism $\phi : G \to S$.

Our rewriting algorithm in S uses the obvious method of writing $g \in S$ as a word in the transpositions $(i, i+1)$ for $1 \leq i \leq n-1$, which can be used to compute images under ϕ^{-1}.

It remains to devise a method of finding $\phi(g)$ for arbitrary $g \in G$.

For this purpose, we compute and store the conjugates $t_i := t^{u^{i-1}}$ for $2 \leq i \leq n$, which satisfy

$$t_i \mapsto (i, i+1) \ (1 \leq i < n), \quad t_n \mapsto (n, 1).$$

Note that it is straightforward to compute and store SLPs over the given generating set X of G for the elements t_i and u.

To compute $\phi(g)$, we calculate the image $i^{\phi(g)}$ for each i.

Since $t_i^g \mapsto (i^{\phi(g)}, (i+1)^{\phi(g)})$, it is sufficient to identify $a_i, b_i \in \{1, \ldots, n\}$ such that $t_i^g \mapsto (a_i, b_i)$.

This can be done by using the fact that $|\{a_i, b_i\} \cap (j, j+1)|$ is equal to 1 if and only t_i^g and t_j do not commute, and is equal to 2 if and only if $t_i^g = t_j$.

Example

Suppose that
$\phi(g) = (1,5,10,11)(2,6,16,13,7,3)(4,19)(8,12,17,14,20,18,15)$
but we do not know that yet!.

We can check that $t_1^g = t_5$, and deduce that $\{1,2\} \mapsto \{5,6\}$.

We check that t_2^g commutes with t_i if and only if $i \notin \{1,2,5,6\}$, and deduce that $t_2^g \mapsto (2,6)$ and hence $\{2,3\} \mapsto \{2,6\}$.

So we have $1^{\phi(g)} = 5$, $2^{\phi(g)} = 6$, $3^{\phi(g)} = 2$, and the other images are calculated similarly.

Constructive recognition of classical groups

Three types of constructive recognition methods have been developed or are under development for the classical groups.

(i) **White box**: for classical groups in their natural representation.

(ii) **Grey box**: for classical groups over a field $\mathbb{F}_q$ that are not in their natural representation, but are given as a matrix groups over a field of the same characteristic as $\mathbb{F}_q$.

(iii) **Black box**: for classical groups over a field $\mathbb{F}_q$ that are given either as permutation groups or as matrix groups over a field of different characteristic from $\mathbb{F}_q$.

For the white box case, we only need rewriting algorithms, and these have been described in the PhD thesis of **Elliot Costi** [10], and implemented by him in Magma..

In the grey box case, we need separate methods for the lower dimensional representations, such as the exterior and symmetric squares of the natural module. These have been developed recently in the PhD thesis of **Brian Corr** [9]. They have been implemented in GAP and are currently being implemented in Magma.

Higher dimensional examples are treated as black box (at least until separate methods are implemented).

The black box case occurs in practice only for classical groups of moderately small dimension (since otherwise the permutation or matrix degree of the representation would be impracticably large).

There is a lengthy treatise by **Kantor and Seress** [19] describing black box algorithms for classical groups, with complete complexity analyses, which have been partially implemented by **Peter Brooksbank**.

They have the weakness that, for classical groups over $\mathbb{F}_q$, there is a factor of q in their complexity, where $\log q$ would be more desirable. They also use discrete logs.

In more recent versions and implementations (see **Conder, Leedham-Green and E.A. O'Brien** [8]), the aim is to reduce the problem polynomially to the black box recognition of $\mathrm{SL}(2, q)$, and then to concentrate separately on that case.

In odd characteristic, an alternative method, due to **Alex Ryba** [15], can be used to recursively reduce the problem to the black box recognition of three involution centralizers. (There is a fast Monte Carlo algorithm due to **John Bray** for computing a probable generating set for an involution centralizer in a black box group, which can be proved to work effectively in classical groups of odd characteristic.)

Constructive recognition of sporadic groups

Robert Wilson et al have defined **standard generators** of the sporadic (and some other) simple groups. These can be found online in [26].

Example

Standard generators of M_{12} are a, b where a is in class 2B, b is in class 3B and ab has order 11.

Black-box algorithms are also provided for finding the standard generators. For example, in M_{12}:

- Find a in class 2B as square of element of order 4;
- Find x in class 2A as fifth power of element of order 10;
- Find b' in class 3B as xx^g for some $g \in G$.
- Find $b = b'^g$ such that $o(ab) = 11$.

For rewriting, BSGS methods are used in the smaller groups and the Ryba method can be used in many of the other cases.

Summing Up

Aschbacher's theorem classifies matrix groups over finite fields into nine types: **C1, . . ., C8, C9=$\mathcal{S}$**.

The first seven of these, **C1–C7**, have associated directly computable homomorphisms onto 'smaller' groups. We have discussed algorithms to recognise matrix groups of these types, and to set up the associated homomorphisms.

These algorithms can be used to reduce the analysis of the matrix group to the study of two smaller groups, the kernel and the image of the homomorphism.

Groups of types **C8** and **C9** are almost simple module the scalar subgroup, and we have discussed algorithms to identify the simple composition factor and to constructively recognise the almost simple quotient.

In practice we may use other directly computable homomorphisms, such as the determinant map, to achieve further reductions; to reduce $\mathrm{GL}_n(q)$ to $\mathrm{SL}_n(q)$, for example.

Section 7

The composition tree algorithm

The composition tree algorithm

We now proceed to describe the **composition tree algorithm** for computing in large (permutation and) matrix groups, which puts together the techniques discussed so far.

We shall be referring here to the version described in the paper by **Bäärnhielm, Holt, Leedham-Green and O'Brien** [2] that is implemented in Magma, but there is an alternative version, described by **Neunhöffer and Seress** in [23], that is available in GAP as the recog package.

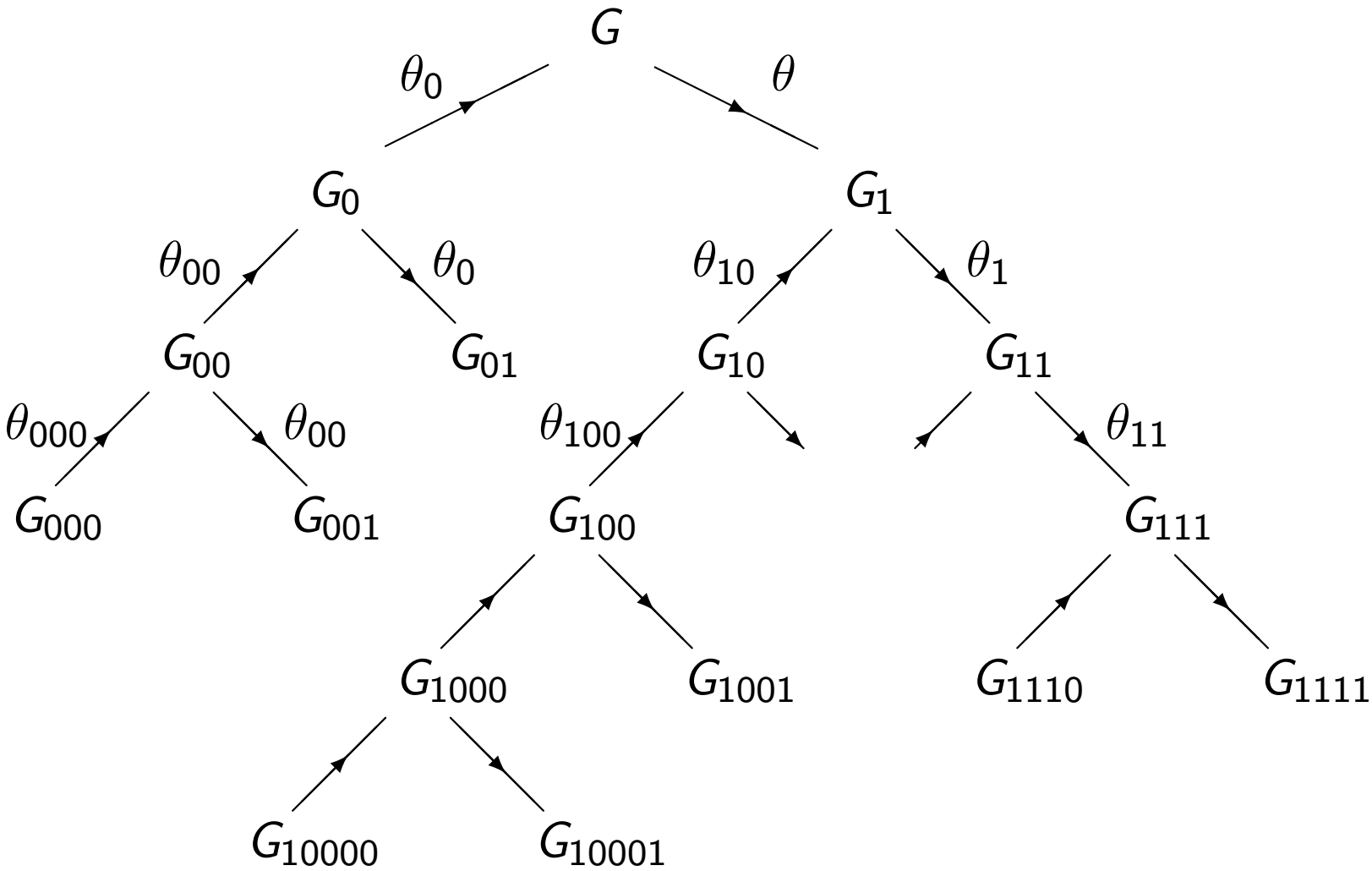
A **composition tree** for a group G has groups as its nodes, with G as the root node. A non-leaf node H has two descendants, H_0 and H_1, with an exact sequence

$$1 \longrightarrow H_0 \overset{\theta_0}{\longrightarrow} H \overset{\theta}{\longrightarrow} H_1 \longrightarrow 1$$

where θ is required to be a directly computable homomorphism.

The map θ_0 is usually just an embedding with $H_0 = \ker \theta$.

The layout of a composition tree



Ideally a leaf node should be simple, in which case a composition tree is theoretically equivalent to a composition series of the root group G.

In the implementations, the leaf nodes are not always simple: they may be arbitrary cyclic or elementary abelian groups or 'nearly' simple groups such as $\mathbf{S}_n$ or $\mathrm{SL}_n(q)$, for which constructive recognition algorithms are available, and so we can easily compute their composition series if required.

We require also that the composition tree algorithm should supply a rewriting algorithm for each node group H in terms of its given generating set (which incorporates a membership testing algorithm when $H \leq \mathrm{Sym}(n)$ or $H \leq \mathrm{GL}_n(q)$).

The generators of $H_1 = \theta_1(H)$ are defined to be the images under θ_1 of the generators of H. We discuss the choice of generators of $H_0 \cong \ker\theta$ below.

For the nonabelian leaf nodes, the rewriting algorithm is provided by the constructive recognition algorithm.

Computing the kernel

We now discuss how to find generators of H_0 for the non-leaf nodes or, equivalently, of the isomorphic group $\ker\theta$.
(As remarked earlier, $\theta_0 : H_0 \to H$ is usually an embedding.)

Except in easy situations, we choose a collection Y of random elements of $\ker\theta$ and check later that they really generate $\ker\theta$.

To do this, we first apply the composition tree algorithm recursively to $H_1 = \langle\theta(X)\rangle$, where $H = \langle X\rangle$.

We can then rewrite elements of H_1 as SLPs over $\theta(X)$, and we the following easy lemma to find random elements of $\ker\theta$.

Lemma

If h is a random element of H, then $h^{-1}Eval(\theta^{-1}(\sigma))$ is a random element of $\ker\theta$, where σ is an SLP for $\theta(h)$ over $\theta(X)$.

We next apply **CompositionTree** recursively to $H_0 = \langle \theta_0^{-1}(Y) \rangle$, which provides us with a membership test for H_0 and enables to rewrite elements of H_0 as SLPs over Y.

Since we may not have chosen enough elements Y to generate $\ker\theta$, we next test some further random elements of $\ker\theta$ for membership of H_0.

Note that if $\ker\theta \neq \langle Y \rangle$ then the probability of a random element of $\ker\theta$ lying in $\langle Y \rangle$ is at most $1/2$, so by testing 20 random elements, say, we can make the failure probability very small.

If a membership test fails, then we add further elements to Y and try again (which means applying **CompositionTree** again).

Otherwise we assume that $H_0 = \langle \theta_0^{-1}(Y) \rangle$. We then combine the rewriting algorithms for H_1 and H_0 to give a rewriting algorithm for H over $Y \cup X$:

for $h \in H$, first rewrite $\theta(h)$ as SLP σ over $\theta(X)$, then rewrite $h^{-1}\mathrm{Eval}(\theta^{-1}(\sigma))$ as SLP τ over Y to give SLP $\tau\sigma$ for h over $Y \cup X$.

Since the elements Y were themselves defined as SLPs over X, we now have the required rewriting algorithm for $H = \langle X \rangle$.

How large should the set Y be?

Theoretical results tell us that the maximum number of generators for $H_0 \leq \mathrm{GL}_n(q)$ with $q = p^e$ is:

- $O(n^2 e)$ when $O_p(H_0) \neq 1$;
- $O(n)$ when $O_p(H_0) = 1$;
- $O(\log n)$ when H_0 is known to be a subnormal subgroup of an irreducible primitive matrix group.

The implementation is designed so that the first situation, $O_p(H_0) \neq 1$, happens only in the first reduction applied to G, when $H_0 = O_p(G)$. So we have at least isolated the most difficult case.

We also generally know when the third situation applies, so we can make use of that result.

In the second situation, examples of H_0 that really require $\Theta(n)$ generators occur relatively rarely in practice, and experiments indicate that we get the fastest average performance by choosing Y to be small initially ($|Y| = 6$ for example), and then doubling the size if it transpires that $\langle Y \rangle \neq \ker \theta$.

We can now present a complete summary of a version of **CompositionTree** that uses Aschbacher's Theorem.

Algorithm: CompositionTree

Input: $G = \langle X \rangle \leq \mathrm{GL}_n(q)$.

1. Test whether G is of one of the geometric types C1–C7.
2. If so then call ProcessNonLeafNode on G.
   Otherwise call ProcessLeafNode on G.

It can happen that G is of geometric type but all of the tests fail. In some cases, such as when G is of type C2 and the action on blocks has trivial kernel, G may be also of Type C9, and the failure of the test is unfortunate, but not fatal.

There is a small probability that all of the tests fail, but G is not of type C8 or C9, in which case ProcessLeafNode will fail.

Algorithm: ProcessNonLeafNode

Input: $G = \langle X \rangle \leq \mathrm{GL}_n(q)$ of Type C1– C7.

1. Define the associated directly computable homomorphism $\theta : G \to H$.
2. If H is abelian or a permutation group, then use the appropriate algorithms to compute a composition tree for H.
   Otherwise (if G is a matrix group) call CompositionTree on $H = \langle \theta(X) \rangle$.
3. Compute a probable generating set Y of $\ker \theta$.
4. Call CompositionTree on $K := \langle Y \rangle$.
5. Test whether a collection of random elements of $\ker \theta$ all lie in K.
   If so, then define the rewriting algorithm for G and stop.
   If not, then return to Step 3 and adjoin further generators of Y.

Remark: If the test in Step 5 fails, then it may be necessary to return to a kernel higher up the tree and add further generators there.

Algorithm: ProcessLeafNode

Input: $G = \langle X \rangle \leq \mathrm{GL}_n(q)$ of Type C8 or C9.

1. Test whether G is of Type C8. If so, call the white-box constructive recognition procedure on G.
2. If not, then attempt to recognize the isomorphism type of the nonabelian simple group $G^\infty/Z(G^\infty)$.
   If this fails, then report failure.
3. Call the appropriate grey- or white-box constructive recognition procedure for G.

Step 3 may also fail. (For example, we may have incorrectly identified $G^\infty/Z(G^\infty)$, which might not be simple.) In that case we again report failure.

In case of failure, we can try running CompositionTree again, devoting more effort to looking for geometric decompositions of the groups in the tree.

Verification

Because of the built in checks, provided that implementation parameters are chosed reasonably sensibly, it is extremely rare for CompositionTree to terminate normally but return an incorrect answer.

However, provided that presentations are available for the leaf groups on their standard generators, the correctness can optionally be formally verified retrospectively.

This process is essentially the same as in permutation groups, and has been discussed alsready in Alexander's lectures.

To do this, we use standard results from the theory of group presentations to construct a presentation of a group H with a normal subgroup H_0 from presentations of H_0 and H/H_0. This allows us to construct presentations of all of the node groups in the tree, ending with G itself.

If this process completes successfully, then we have verified correctness of the tree.

Section 7

Structural computations using CompositionTree

Structural computations using CompositionTree

Suppose that we have computed a composition tree $\mathcal{T}$ for $G \leq \mathrm{GL}_n(q)$.

Unfortunately, for a subgroup $H < G$, unless H happens to be one of the node groups of $\mathcal{T}$, there appears to be no effective method of using $\mathcal{T}$ to compute a composition tree for H. So we are forced to treat H as an unrelated group.

On the other hand, since any base for G is a base for H, we can use a BSGS for G to compute one for H, so this is a good reason for using a BSGS whenever possible.

Given a composition tree (or a BSGS) for H, we can test H for **normality** in G.

So there are straightforward algorithms for computing **normal closures** of subsets of G, and hence for computing the terms in the **derived** and **lower central series** of G.

The solvable radical approach

There is a family of algorithms for computing in finite groups G, which have the following general structure.

1. Compute the solvable radical $L := O\infty(G)$ of G.
2. Solve the problem in G/L using known properties of the nonabelian simple direct factors of $\mathrm{Soc}(G/L)$.
3. Solve the problem in G using linear algebra to lift the solution through the elementary abelian layers of L.

They have been used mainly in permutation groups, matrix groups with BSGS, and solvable groups defined by a polycyclic (PC) presentation (with $L = G$), but recent progress has been made in their application to matrix groups with composition tree.

The series was also used by **Babai and Beals** in their algorithms for black box groups described in [3].

Examples of problems for which this approach has proved useful are:

- Find (representatives of) the **conjugacy classes** of G.
- Find (representatives of the conjugacy classes of) **subgroups** of G; or just the **maximal subgroups**, etc.
- Compute the **automorphism group** of G, or test two groups for **isomorphism**.

In some cases, such as finding conjugacy classes, the algorithm was a generalization of an existing method for solvable groups defined by a PC-presentation.

In Step 2 (above), we first compute the socle $\mathrm{Soc}(G/L) = M/L$ of G/L, which is a direct product of nonabelian simple groups S_i $(1 \leq i \leq m)$.

The groups S_i are permuted under the action of conjugation in G/L, and the kernel of this permutation action was denoted by PKer by Babai and Beals. So we have a series of characteristic subgroups:

$$1 \leq L \leq M \leq \mathrm{PKer} \leq G$$

where PKer/M and G/PKer is isomorphic to subgroups of $\times_{i=1}^{m} \mathrm{Out} S_i$ and the symmetric group $\mathbf{S}_m$, respectively.

For groups within the range of practical computation m is reasonably small, and the groups $\mathrm{Out} S_i$ are also typically small (and solvable), so G/M is generally of manageable size.

For computational purposes, the difficult section of the group is usually M/L, which can be very large. But, since the simple groups S_i have been constructively recognized by CompositionTree, we can hope to use theoretical results to "write down" the solution of the problem in S_i.

For example, we can write down the maximal subgroups of (almost) simple classical groups of dimensions up to 12.

Finding L and M

Before we can apply these methods to a matrix group with a composition tree $\mathcal{T}$, we first need to find the subgroups

$$L = O\infty(G) \quad \text{and} \quad M \quad \text{with} \quad M/L = \mathrm{Soc}(G/L).$$

We can construct a composition series for G from T, but we need to determine which of its cyclic factors lie in L and which of its nonabelian factors lie in M.

In fact, both of those problems reduce to the following.

Problem (SwapFactors)

Let $1 \to N \to G \to K \to 1$ be a short exact sequence of groups in which N and K are simple. Decide whether N has a normal complement C in G and, if so, find C together with an effective homomorphism $G \to N$ with kernel C.

If we can solve this problem, then we can rearrange the terms in the composition series to make it pass through L and M.

Solving SwapFactors

This is easy if G and K are both cyclic of prime order.

If N is cyclic and K is nonabelian then, if the complement C exists, we have $C = [G, G]$, so this case is also not hard.

When N is nonabelian simple, SwapFactors reduces, in turn, to:

Problem (IdentifyAutomorphism)

Given a finite nonabelian simple group N and an effectively computable automorphism α of N, identify the image of α in $\mathrm{Out}N$ and, if $\alpha \in \mathrm{Inn}N$, then find $g \in N$ such that α is conjugation by g.

For $N = \mathbf{A}_n$, we can use the black box methods discussed earlier to identify α as an element of $\mathbf{S}_n$.

For sporadic groups and exceptional groups of Lie Type, we have no special methods at present, so we have to use BSGS based techniques.

IdentifyAutomorphism for classical groups

It is a standard result that an automorphism α of a classical simple group G can be written as a product $\iota\delta\phi\gamma$, where ι, δ, ϕ and γ are respectively **inner**, **diagonal**, **field** and **graph** automorphisms of G.

Let $c(g)$ be the characteristic polynomial of $g \in G$. Then

- $c(\iota(g)) = c(\delta(g)) = c(g)$
- $c(\phi(g)) = \phi(c(g))$
- For γ of order 2 in the linear case, $c(\gamma(g)) = c(g^{-1})$.

So, by calculating $c(g)$ and $c(\alpha(g))$ for a small number of random $g \in G$, we can identify ϕ and (in the linear case) γ.

When $\alpha = \iota\delta$, α is induced by conjugation by an element x in the linear group $\mathrm{GL}_d(q)$ containing G.

So, if M is the natural d-dimensional module over $\mathbb{F}_q$ for G, then M and M^α are isomorphic and (up to a scalar multiple), x is the matrix inducing this isomorphism. We can find x using **MeatAxe** methods.

Further algorithms

So we can find a composition series that passes through the characteristic subgroups in the series

$$1 \leq L \leq M \leq \mathrm{PKer} \leq G.$$

We can use our rewriting algorithms to compute a PC-presentation of the solvable radical L. In fact, we compute an isomorphism ϕ from L to a PC-group.

Structural computations in G that now become feasible include:

- Find a chief series of G.
- Find the centre $Z(G)$ of G.
- Find the groups in the upper central series of G.
- Find the Fitting subgroup of G.
- Find Sylow subgroups of G.

Example

In the first lecture, we used used the algorithm of Detinko, Flannery and O'Brien [12] to find an isomorphic copy of $G \cong 6.\mathrm{Suz} < \mathrm{GL}_{12}(\mathbb{Q}(w))$ in $\mathrm{GL}_{12}(25)$. (The order of G is $2690072985600 = 2^{14}.3^8.5^2.7.11.13$.)

Using the algorithms above, we can find a chief series of this isomorphic copy, its centre, and all of its Sylow subgroups in less than 4 seconds.

It was not possible to carry out such computations directly in the original group G.

But the rewriting algorithms resulting from the application of CompositionTree to the image of G in $\mathrm{GL}_{12}(25)$ enable us to lift the results back to G if required.

Compromised algorithms

For most other algorithms that use the above characteristic series, we need first to carry out the computation in G/L, so we need a nice representation of G/L.

This does not seem easy in general, but may be feasible in important special cases, such as when M/L is a classical group, so G/L is almost simple.

There are many interesting examples, including some of the maximal subgroups of the finite simple groups, for which we cannot readily compute a BSGS of the whole group G, but we can compute a manageable permutation representation

$$\rho : G \to P \text{ with } \ker \rho = L, \text{ and } P \cong G/L.$$

We can then:

- Find **(maximal) subgroups** of G.
- Compute **centralizers** or **normalizers** of elements or subgroups of G.
- Test elements or subgroups of G for **conjugacy** in G.

The centralizer, normalizer, and conjugacy testing algorithms, together with the facility to find representatives of conjugacy classes of elements in G are described by **Alexander Hulpke** in [18], who has implemented them in the GAP **recog** package.

The conjugacy class representative program, in particular, requires a large number of applications of the isomorphism ϕ from L to a PC-group, and using the the composition tree mechanism for these evaluations turns out to be prohibitively slow in larger examples, such as

$$2^{9+16}.S_8(2) \leq \mathrm{GL}_{394}(2) \quad \text{or} \quad 3^{1+12}.2.\mathrm{Suz}.2 \leq \mathrm{GL}_{78}(3).$$

In many such examples, it is possible to find a BSGS for L, which makes the evaluation of ϕ faster by a factor of 10.

CompositionTree was implemented in Magma by **Henrik Bäärnhielm** and **Eamonn O'Brien**, using code written by many people over the past 20 years.

The algorithms that use **CompositionTree** to carry our further structural computations are implemented in Magma as the **LMG** (large matrix group) functions.

Before embarking on any calculation with a group $G \leq \mathrm{GL}_n(q)$, these functions attempt briefly to compute a BSGS for G and, if this succeeds, then they use the existing Magma BSGS based methods for all further calculations with G.

This is not always a good idea, and the user can control the amount of effort devoted to finding a BSGS, including setting this to 0, which would mean definitely use **CompositionTree**.

THE END

Bibliography

Disclaimer: This is far from being a complete list of relevant papers, and is just included as a representative sample. There is a more complete list in the paper [2].

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

[2] H. Bäärnhielm, D.F. Holt, C.R. Leedham-Green, E.A. O'Brien. A practical model for computation with matrix groups. To appear in *J. Symbolic Computation*.

[3] L. Babai and R. Beals. A polynomial-time theory of black box groups, I. In *Groups St. Andrews 1997 in Bath, I*, volume 260 of *London Math. Soc. Lecture Note Ser.*, pages 30–64. Cambridge Univ. Press, Cambridge, 1999.

[4] S. Bratus and 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**, 33–57, 2000.

[5] G. Butler. The Schreier algorithm for matrix groups. In SYMSAC '76, *Proc. ACM Sympos. Symbolic and Algebraic Computation*, pages 167–170, New York, 1976, Association for Computing Machinery, 1976.

[6] F. Celler and C.R. Leedham-Green. Calculating the order of an invertible matrix. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 55–60. Amer. Math. Soc., Providence, RI, 1997.

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

[8] M.D.E. Conder, C.R. Leedham-Green, and E.A. O'Brien. Constructive recognition of $\mathrm{PSL}(2, q)$. *Trans. Amer. Math. Soc.*, **358**, 1203–1221, 2006.

[9] Brian Corr. *Estimation and computation with matrices over finite field*. PhD thesis, University of Western Australia, 2013.

[10] Elliot Costi. *Constructive membership testing in classical groups*. PhD thesis, Queen Mary, University of London, 2009.

[11] The Cunningham Project.
`http://en.wikipedia.org/wiki/Cunningham_project`

[12] A.S. Detinko, D.L. Flannery and E.A. O'Brien. Recognizing finite matrix groups over infinite fields. *J. Symb. Comput.*, **50**, 100–109, 2013.

[13] A.S. Detinko, D.L. Flannery and E.A. O'Brien. Algorithms for the Tits alternative and related problems. *J. Algebra* **344**, 397–406, 2011.

[14] S.P. Glasby, C.R. Leedham-Green and E.A. O'Brien. Writing projective representations over subfields. *J. Algebra*, **295**, 51–61, 2006.

[15] P.E. Holmes, S.A. Linton, E.A. O'Brien, A.J.E. Ryba and R.A. Wilson. *Constructive membership in black-box groups*, J. Group Theory **11**, 747–763, 2008.

[16] D.F. Holt, C.R. Leedham-Green, E.A. O'Brien and S. Rees. Computing matrix group decompositions with respect to a normal subgroup. *J. Algebra*, **184**, 818–838, 1996.

[17] D.F. Holt and S. Rees. Testing modules for irreducibility. *J. Aust. Math. Soc. Ser. A*, **57**, 1–16, 1994.

[18] A. Hulpke. Computing Conjugacy Classes of Elements in Matrix Groups. To appear in *J. Algebra*.

[19] W.M. Kantor, Á. Seress. *Black box classical groups*, Mem. Amer. Math. Soc. **149** (2001), no.708, viii+168.

[20] Peter Kleidman and Martin Liebeck. *The subgroup structure of the finite classical groups.* London Math. Soc. Lecture Note Ser., 129. *Cambridge University Press, Cambridge*, 1990.

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

[22] P.M. Neumann and C.E. Praeger. A recognition algorithm for special linear groups. *Proc. Lon. Math. Soc.* **65**, 555–603, 1992.

[23] Max Neunhöffer and Á, Seress. A data structure for a uniform approach to computations with finite groups. In ISSAC 2006, pages 254–261. ACM, New York, 2006.

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

[25] Robert A. Wilson. *The Finite Simple groups*, Graduate Texts in Mathematics **251**, Springer, 2009.

[26] R.A. Wilson, P.G. Walsh, J. Tripp, I.A.I. Suleiman, R.,A. Parker, S.P. Norton, S.J. Nickerson, S.A. Linton, J.N. Bray and R.,A. Abbott. ATLAS of Finite Group Representations.
`http://brauer.maths.qmul.ac.uk/Atlas/v3/`