Computing with matrix groups

Peter Brooksbank

Bucknell University

Linear Algebra and Matrix Theory:
connections, applications and computations
NUI Galway (December 4, 2012)

Sims' Legacy

- The fundamental techniques for computing with permutation groups were developed by Charles Sims in the early 1970's.
- Construction of sporadic simple groups Ly and B.
- Basic data structure: stabilizer chain.
  Let $G = \langle X \rangle \leqslant S_n$ be given. For $i = 1, \ldots, n$, let $G^{(i)}$ be the subgroup of G fixing $1, \ldots, i-1$. Then

  $$G = G^{(1)} \geqslant G^{(2)} \geqslant \ldots \geqslant G^{(n)} = 1,$$

  and $[G^{(i)} : G^{(i+1)}] \leqslant n$.

Basic Polynomial Time Theory

- That Sims' techniques were polynomial time was established by Furst, Hopcroft & Luks (1981).
- Given $G = \langle X \rangle \leqslant S_n$, one can:
  - Compute orbits of G (and decide if it acts transitively);
  - Compute a block system for G (and decide if it's primitive);
  - Determine $|G|$;
  - Given $x \in S_n$, decide whether $x \in G$;
  - Find the derived series $G \geqslant G' \geqslant (G')' \geqslant \ldots$ (test solubility);
  - Find the lower central series (test nilpotence).

Basic Polynomial Time Theory

- That Sims' techniques were polynomial time was established by Furst, Hopcroft & Luks (1981).
- Given $G = \langle X \rangle \leqslant S_n$, one can:
  - Compute orbits of G (and decide if it acts transitively);
  - Compute a block system for G (and decide if it's primitive);
  - Determine $|G|$;
  - Given $x \in S_n$, decide whether $x \in G$;
  - Find the derived series $G \geqslant G' \geqslant (G')' \geqslant \ldots$ (test solubility);
  - Find the lower central series (test nilpotence).
- "Permutation groups and polynomial-time computation", E.M. Luks, DIMACS series, 1993.

Deeper Results

Other permutation group problems are shown to be in polynomial time using deeper results about finite groups.

1. As demonstrated by Luks (1987), a composition series

$$1 = N_1 \trianglelefteq N_2 \trianglelefteq \ldots \trianglelefteq N_t = G$$

can be constructed using the classification of primitive permutation groups of O'Nan & Scott.

Deeper Results

Other permutation group problems are shown to be in polynomial time using deeper results about finite groups.

1. As demonstrated by Luks (1987), a composition series

$$1 = N_1 \trianglelefteq N_2 \trianglelefteq \ldots \trianglelefteq N_t = G$$

can be constructed using the classification of primitive permutation groups of O'Nan & Scott.

2. Kantor (1985) showed how to construct Sylow subgroups. Relies on the Classification of Finite Simple Groups (CFSG).

"Simple groups in computational group theory", W.M. Kantor, Proc. ICM (Berlin, 1998).

P≠NP?

Motivation for the algorithmic study of permutation groups came from two sources:

1. The desire to put Sims' methods on a solid theoretical basis;
2. The connection to the Graph Isomorphism Problem:
   Given two finite graphs Γ_1 *and* Γ_2*, decide whether* $\Gamma_1 \cong \Gamma_2$.

Despite the many different combinatorial attacks on this problem, by far the best result fundamentally uses permutation group algorithms:

"Isomorphism of graphs of bounded valence can be tested in polynomial time", E.M. Luks, J. Comp. Sys. Sci (1982).

Practical Range

- The machinery for computing with permutation groups in GAP and MAGMA is incredibly efficient.
- One can compute effectively with $G \leqslant S_n$ for $n \approx 10^{10}$.
- Even problems for which no polynomial-time algorithm is known can still be solved extremely quickly using these systems.
- The algorithmic theory of permutation groups is on solid ground both from a practical and a theoretical viewpoint.

What's The Problem?

Let $G = \langle X \rangle \leqslant \mathrm{GL}(d, \mathbb{F}_q)$. Then clearly G is a group of permutations of the $q^d - 1$ nonzero vectors in $\mathbb{F}_q^d$. So why not view G as a subgroup of S_{q^d-1} and use permutation group methods to study G?

What's The Problem?

Let $G = \langle X \rangle \leqslant \mathrm{GL}(d, \mathbb{F}_q)$. Then clearly G is a group of permutations of the $q^d - 1$ nonzero vectors in $\mathbb{F}_q^d$. So why not view G as a subgroup of S_{q^d-1} and use permutation group methods to study G?

- Until fairly recently this is precisely what GAP and MAGMA did.
- Exponential blow up in input length: $d^2 \log q$ versus q^d.
- G may not have a subgroup of polynomially bounded index.
- This limits the practical range to something like $\mathrm{GL}(8, \mathbb{F}_5)$.

What's The Problem?

Let $G = \langle X \rangle \leqslant \mathrm{GL}(d, \mathbb{F}_q)$. Then clearly G is a group of permutations of the $q^d - 1$ nonzero vectors in $\mathbb{F}_q^d$. So why not view G as a subgroup of S_{q^d-1} and use permutation group methods to study G?

- Until fairly recently this is precisely what GAP and MAGMA did.
- Exponential blow up in input length: $d^2 \log q$ versus q^d.
- G may not have a subgroup of polynomially bounded index.
- This limits the practical range to something like $\mathrm{GL}(8, \mathbb{F}_5)$.

Matrix representations are very concise!

Neubüser's Question

At the Oberwolfach meeting on Computational Group Theory in 1988, Neubüser asked for

> *...an efficient algorithm to decide whether a given group* $G = \langle X \rangle \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ *contains* $\mathrm{SL}(d, \mathbb{F}_q)$.

Neubüser's Question

At the Oberwolfach meeting on Computational Group Theory in 1988, Neubüser asked for

> *...an efficient algorithm to decide whether a given group* $G = \langle X \rangle \leqslant \mathrm{GL}(d, \mathbb{F}_q)$ *contains* $\mathrm{SL}(d, \mathbb{F}_q)$.

In 1991 Neumann & Praeger supplied such an algorithm:

- Identify certain elements which abound in $\mathrm{SL}(d, \mathbb{F}_q)$, but which were unlikely to be found in groups not containing $\mathrm{SL}(d, \mathbb{F}_q)$.
- Results in a 1-sided Monte Carlo algorithm.
- They showed that progress can be made with matrix groups.
- Their result demonstrated the power of randomization.

The Matrix Group Project

The goal is to devise algorithms which, given any matrix group $G = \langle X \rangle \leqslant \mathrm{GL}(d, \mathbb{F}_q)$, do the following:

- Find $|G|$;
- Determine the structure of G via a composition series;
- Set up data structures to compute effectively with G.

Two different strategies emerged:

- geometric approach which aims to find a composition tree; and
- black box approach which aims to construct the series

$$1 \leqslant O_\infty(G) \leqslant \mathrm{Soc}^*(G) \leqslant \mathrm{PKer}(G) \leqslant G.$$

Aschbacher's Theorem

"On the maximal subgroups of the finite classical groups",
M. Aschbacher, Invent. Math. (1984).

For our purpose, we summarize Aschbacher's theorem as follows:

Theorem

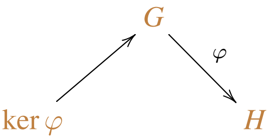
For a maximal subgroup G of $\mathrm{GL}(d, \mathbb{F}_q)$, one of the following holds:

1. *G preserves one of seven types of geometric structure on $\mathbb{F}_q^d$;*
2. *G normalizes a classical group in its natural representation; or*
3. *G is almost simple modulo scalars.*

In part 1, for example, G might stabilize a subspace of $\mathbb{F}_q^d$ ... which one could test using the Meataxe algorithm described yesterday.

Composition Tree

A divide-and-conquer strategy based on Aschbacher.

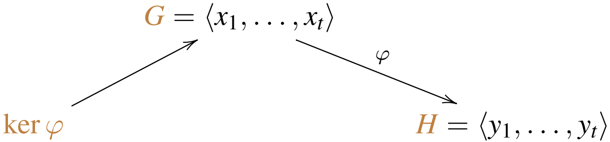


- $\varphi\colon G \to H$ is an "Aschbacher reduction", where H is the group induced by G on a suitable geometric structure.
- Recursively find comp. trees for subtrees rooted at H and $\ker \varphi$.
- Glue together to obtain comp. tree for G.
- Recursion bottoms out with almost simples and classicals.

"A new model for computation with matrix groups",
Bäanhielm, Holt, Leedham-Green, O'Brien, preprint (2011).

Generating Kernels & Verification

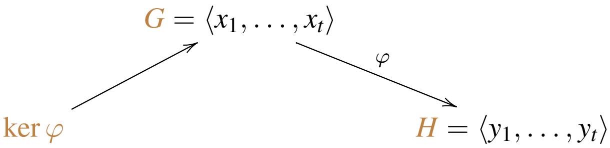
Suppose we have constructed a composition tree for H.



This allows us to write any given element $h \in H$ as a word in the $y_i = x_i\varphi$. We now wish to find generators for $\ker \varphi$.

Generating Kernels & Verification

Suppose we have constructed a composition tree for H.

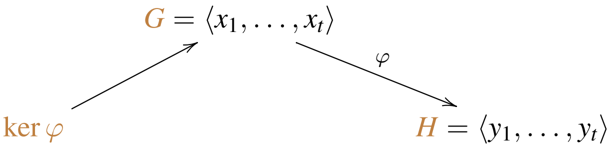


This allows us to write any given element $h \in H$ as a word in the $y_i = x_i\varphi$. We now wish to find generators for $\ker \varphi$.

1. Generate a random element $g \in G$.

Generating Kernels & Verification

Suppose we have constructed a composition tree for H.

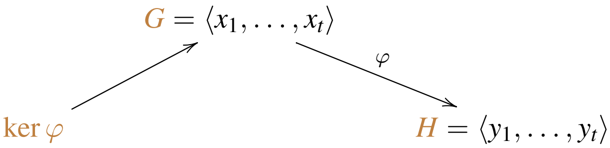


This allows us to write any given element $h \in H$ as a word in the $y_i = x_i\varphi$. We now wish to find generators for $\ker \varphi$.

1. Generate a random element $g \in G$.
2. Compute $h = g\varphi \in H$ and write $h = w(y_1, \ldots, y_t)$.

Generating Kernels & Verification

Suppose we have constructed a composition tree for H.

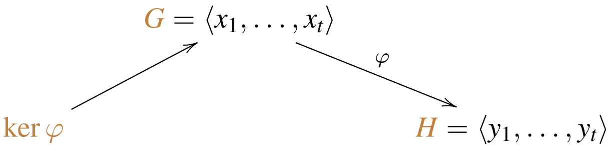


This allows us to write any given element $h \in H$ as a word in the $y_i = x_i\varphi$. We now wish to find generators for $\ker \varphi$.

1. Generate a random element $g \in G$.
2. Compute $h = g\varphi \in H$ and write $h = w(y_1, \ldots, y_t)$.
3. Evaluate $x = w(x_1, \ldots, x_t) \in G$.

Generating Kernels & Verification

Suppose we have constructed a composition tree for H.

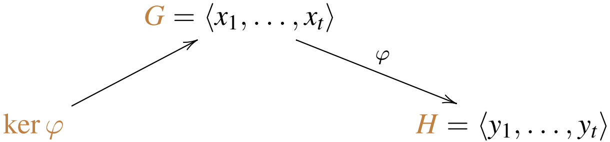


This allows us to write any given element $h \in H$ as a word in the $y_i = x_i\varphi$. We now wish to find generators for $\ker \varphi$.

1. Generate a random element $g \in G$.
2. Compute $h = g\varphi \in H$ and write $h = w(y_1, \ldots, y_t)$.
3. Evaluate $x = w(x_1, \ldots, x_t) \in G$.
4. Add gx^{-1} to a generating set for $\ker \varphi$.

Generating Kernels & Verification

Suppose we have constructed a composition tree for H.



This allows us to write any given element $h \in H$ as a word in the $y_i = x_i\varphi$. We now wish to find generators for $\ker \varphi$.

1. Generate a random element $g \in G$.
2. Compute $h = g\varphi \in H$ and write $h = w(y_1, \ldots, y_t)$.
3. Evaluate $x = w(x_1, \ldots, x_t) \in G$.
4. Add gx^{-1} to a generating set for $\ker \varphi$.

Similarly, one constructs a presentation for G from presentations for H and $\ker \varphi$. Hence one can verify the correctness of the entire tree.

Non-constructive Recognition

The success of the entire procedure depends crucially on our ability to compute effectively with the groups at the "leaves" of the tree, namely the almost simple groups, and the classical groups.

Non-constructive Recognition

The success of the entire procedure depends crucially on our ability to compute effectively with the groups at the "leaves" of the tree, namely the almost simple groups, and the classical groups.

The first task is to determine which simple group we have:

Theorem

There is a polynomial-time Monte Carlo algorithm which, given any simple group $G = \langle X \rangle$, returns the "name" of G.

e.g. "$\mathrm{PSL}(3, 11)$" or "$\mathrm{P}\Omega^{+}(8, 3)$".

Non-constructive Recognition

The success of the entire procedure depends crucially on our ability to compute effectively with the groups at the "leaves" of the tree, namely the almost simple groups, and the classical groups.

The first task is to determine which simple group we have:

Theorem

There is a polynomial-time Monte Carlo algorithm which, given any simple group $G = \langle X \rangle$, returns the "name" of G.

e.g. "$\mathrm{PSL}(3, 11)$" or "$\mathrm{P}\Omega^{+}(8, 3)$".

There are many contributors to this result (there are others):

- Neumann & Praeger (1992)
- Niemeyer & Praeger (1998)
- Babai, Kantor, Pálfy, Seress (2002)
- Altseimer & Borovik (2001)

Explicit Isomorphism

Suppose that $G = \langle X \rangle$ is simple of some known type, and let T be the "standard copy" of this simple group. We require:

1. An explicit isomorphism $\varphi\colon G \to T$. This means we need algorithms which:
   - given $g \in G$ compute the image $g\varphi \in T$; and
   - given $t \in T$ compute the pre image $t\varphi^{-1} \in G$.
2. A rewriting algorithm: given $g \in G$, write g as a word in X.

Explicit Isomorphism

Suppose that $G = \langle X \rangle$ is simple of some known type, and let T be the "standard copy" of this simple group. We require:

1. An explicit isomorphism $\varphi\colon G \to T$. This means we need algorithms which:
   - given $g \in G$ compute the image $g\varphi \in T$; and
   - given $t \in T$ compute the pre image $t\varphi^{-1} \in G$.
2. A rewriting algorithm: given $g \in G$, write g as a word in X.

In practice, 1 and 2 are achieved via the same process:

(a) Find a new set Y of generators for G (as words in X) whose image under φ is a "nice" generating set for T.

Explicit Isomorphism

Suppose that $G = \langle X \rangle$ is simple of some known type, and let T be the "standard copy" of this simple group. We require:

1. An explicit isomorphism $\varphi \colon G \to T$. This means we need algorithms which:
   - given $g \in G$ compute the image $g\varphi \in T$; and
   - given $t \in T$ compute the pre image $t\varphi^{-1} \in G$.
2. A rewriting algorithm: given $g \in G$, write g as a word in X.

In practice, 1 and 2 are achieved via the same process:

(a) Find a new set Y of generators for G (as words in X) whose image under φ is a "nice" generating set for T.

(b) Given $g \in G$ write g as a word in Y:
   - compose with words expressing elements of Y in terms of X to get g as a word in X; or
   - evaluate on $Y\varphi$ to compute $g\varphi$.

Rewriting Algorithms

Linear algebra is used heavily here.

- Let $T = \mathrm{SL}(d, \mathbb{F}_q)$.
- Elements of $Y\varphi$ are elementary transvections, e.g.

$$X_{23}(\lambda) = I_4 + E_{23}(\lambda) = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ \cdot & 1 & \lambda & \cdot \\ \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & 1 \end{bmatrix}$$

- Writing $t \in T$ as a word in the $X_{ij}(\lambda)$ is effected by Gaussian elimination.
- Performing the analogous procedure in G is harder, but tractable.
- Constructing the elements of Y is by far the hardest part!

$\mathrm{SL}(2, q)$

Let $G = \mathrm{SL}(2, \mathbb{F}_q)$. The goal is to construct an element conjugate to

$$\begin{bmatrix} 1 & * \\ \cdot & 1 \end{bmatrix}$$

- The proportion of such elements is roughly $\frac{1}{q}$; if q is fairly large (say $q = 2^{31}$), a random search will fail to locate one.

$\mathrm{SL}(2, q)$

Let $G = \mathrm{SL}(2, \mathbb{F}_q)$. The goal is to construct an element conjugate to

$$\begin{bmatrix} 1 & * \\ \cdot & 1 \end{bmatrix}$$

- The proportion of such elements is roughly $\frac{1}{q}$; if q is fairly large (say $q = 2^{31}$), a random search will fail to locate one.
- Roughly half of the elements $\mathrm{SL}(2, \mathbb{F}_q)$ have order dividing $q - 1$ (most of the remaining elements having order dividing $q + 1$), so a random search will produce elements of order $q - 1$.

$\mathrm{SL}(2, q)$

Let $G = \mathrm{SL}(2, \mathbb{F}_q)$. The goal is to construct an element conjugate to

$$\begin{bmatrix} 1 & * \\ \cdot & 1 \end{bmatrix}$$

- The proportion of such elements is roughly $\frac{1}{q}$; if q is fairly large (say $q = 2^{31}$), a random search will fail to locate one.
- Roughly half of the elements $\mathrm{SL}(2, \mathbb{F}_q)$ have order dividing $q - 1$ (most of the remaining elements having order dividing $q + 1$), so a random search will produce elements of order $q - 1$.

Can we use them to construct a transvection?

The Discrete Log Trick

Given $G = \langle X \rangle = \mathrm{SL}(2, \mathbb{F}_q)$.

1. Find $A \in G$ of order $q - 1$, having eigenvectors u and v.

The Discrete Log Trick

Given $G = \langle X \rangle = \mathrm{SL}(2, \mathbb{F}_q)$.

1. Find $A \in G$ of order $q - 1$, having eigenvectors u and v.
2. Let B be a random G-conjugate of A.
   [We may assume A and B share no eigenspace]

The Discrete Log Trick

Given $G = \langle X \rangle = \mathrm{SL}(2, \mathbb{F}_q)$.

1. Find $A \in G$ of order $q-1$, having eigenvectors u and v.
2. Let B be a random G-conjugate of A.
   [We may assume A and B share no eigenspace]
3. Take another random element C of G, and find i such that $B^i C$ fixes $\langle u \rangle$ as follows:
   - choose a basis for $\mathbb{F}_q^2$ such that $B = \begin{bmatrix} a & \cdot \\ \cdot & a^{-1} \end{bmatrix}$;
     relative to this basis, $u = \langle 1, d \rangle$ and $C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$.
   - using discrete logarithms, find i such that

$$a^{2i} = \frac{d^2 c_{21} - d c_{22}}{c_{12} - d c_{11}}.$$

The Discrete Log Trick

Given $G = \langle X \rangle = \mathrm{SL}(2, \mathbb{F}_q)$.

1. Find $A \in G$ of order $q-1$, having eigenvectors u and v.
2. Let B be a random G-conjugate of A.
   [We may assume A and B share no eigenspace]
3. Take another random element C of G, and find i such that $B^i C$ fixes $\langle u \rangle$ as follows:
   - choose a basis for $\mathbb{F}_q^2$ such that $B = \begin{bmatrix} a & \cdot \\ \cdot & a^{-1} \end{bmatrix}$;
     relative to this basis, $u = \langle 1, d \rangle$ and $C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$.
   - using discrete logarithms, find i such that

$$a^{2i} = \frac{d^2 c_{21} - d c_{22}}{c_{12} - d c_{11}}.$$

4. $[A, B^i C]$ is the desired transvection.

Results and Remarks

Theorem

Assuming the availability of an algorithm ("oracle") to handle $\mathrm{SL}(2, \mathbb{F}_q)$*, there are polynomial-time Las Vegas constructive recognition algorithms for all finite classical groups.*

Results and Remarks

Theorem

Assuming the availability of an algorithm ("oracle") to handle $\mathrm{SL}(2, \mathbb{F}_q)$*, there are polynomial-time Las Vegas constructive recognition algorithms for all finite classical groups.*

Natural characteristic

- Odd characteristic: Leedham-Green & O'Brien (2009)
- Even characteristic: ...+ Dietrich & Lübeck (2012+)

Results and Remarks

Theorem

Assuming the availability of an algorithm ("oracle") to handle $\mathrm{SL}(2, \mathbb{F}_q)$*, there are polynomial-time Las Vegas constructive recognition algorithms for all finite classical groups.*

Natural characteristic

- Odd characteristic: Leedham-Green & O'Brien (2009)
- Even characteristic: ...+ Dietrich & Lübeck (2012+)

Black box

- $\mathrm{PSL}(d, \mathbb{F}_q)$: B & Kantor (2001)
- $\mathrm{PSU}(d, \mathbb{F}_q)$: B (2003)
- $\mathrm{P}\Omega^{\epsilon}(d, \mathbb{F}_q)$: B & Kantor (2006)
- $\mathrm{PSp}(d, \mathbb{F}_q)$: B (2008)

Fruits Of The Tree?

- A complete implementation of the composition tree algorithm is now distributed with MAGMA.
- It can be adapted to return the characteristic series

$$1 \leqslant O_\infty(G) \leqslant \mathrm{Soc}^*(G) \leqslant \mathrm{PKer}(G) \leqslant G.$$

- Which computational problems can we hope to solve with it?
    - Conjugacy classes of G.
    - Maximal subgroups of G.
    - $\mathrm{Aut}(G)$.

Infinite Fields

One of the most exciting new directions in computational group theory is the algorithmic study of matrix groups over **infinite fields**.

The most impressive contributions to date are from a collaboration of Detinko, Flannery & O'Brien. They have devised algorithms to:

- decide finiteness; and
- resolve the Tits alternative.