Algorithms for matrix groups

Eamonn O'Brien

University of Auckland

December 2010

Overview

$G = \langle X \rangle \leq \mathrm{GL}(d, R)$ where R is a ring; usually finite field $\mathrm{GF}(q)$

Overview

$G = \langle X \rangle \leq \mathrm{GL}(d, R)$ where R is a ring; usually finite field $\mathrm{GF}(q)$

Goal: efficient algorithms, for their study, which are both theoretically and practically effective.

Why do we care?

Why do we care?

- Modular representation theory: Dickson (1910s), applications to number theory, algebraic groups etc.

Why do we care?

- Modular representation theory: Dickson (1910s), applications to number theory, algebraic groups etc.
- Sporadic simple groups: constructed as irreducible representations over small fields.

Why do we care?

- Modular representation theory: Dickson (1910s), applications to number theory, algebraic groups etc.
- Sporadic simple groups: constructed as irreducible representations over small fields.
  Benson et al. (1982): $J_4 \leq \mathrm{GL}(112, 2)$, order 10^{20}.

Why do we care?

- Modular representation theory: Dickson (1910s), applications to number theory, algebraic groups etc.
- Sporadic simple groups: constructed as irreducible representations over small fields.
  Benson et al. (1982): $J_4 \leq \mathrm{GL}(112, 2)$, order 10^{20}.
- Invariant theory: irreducible representations, Kronecker products, tensor-induced representations.

Why do we care?

- Modular representation theory: Dickson (1910s), applications to number theory, algebraic groups etc.
- Sporadic simple groups: constructed as irreducible representations over small fields.
  Benson et al. (1982): $J_4 \leq \mathrm{GL}(112, 2)$, order 10^{20}.
- Invariant theory: irreducible representations, Kronecker products, tensor-induced representations.
- Energy levels of systems of identical particles: irreducible representations of classical groups

Cost of matrix multiplication

Two $d \times d$ matrices A and B
Cost of $A \times B$ using conventional algorithm is $O(d^3)$.

Cost of matrix multiplication

Two $d \times d$ matrices A and B
Cost of $A \times B$ using conventional algorithm is $O(d^3)$.

Strassen: $O(d^{\log_2(7)})$

Cost of matrix multiplication

Two $d \times d$ matrices A and B
Cost of $A \times B$ using conventional algorithm is $O(d^3)$.

Strassen: $O(d^{\log_2(7)})$

Coppersmith & Winograd (1990): $O(d^{2.37})$

Cost of matrix multiplication

Two $d \times d$ matrices A and B
Cost of $A \times B$ using conventional algorithm is $O(d^3)$.

Strassen: $O(d^{\log_2(7)})$

Coppersmith & Winograd (1990): $O(d^{2.37})$

Where do we notice improvements?

Cost of matrix multiplication

Two $d \times d$ matrices A and B
Cost of $A \times B$ using conventional algorithm is $O(d^3)$.

Strassen: $O(d^{\log_2(7)})$

Coppersmith & Winograd (1990): $O(d^{2.37})$

Where do we notice improvements? Perhaps for $d \geq 100$.

Membership

Given $G \leq \mathrm{GL}(d, \mathbb{Z})$, and $x \in \mathrm{GL}(d, \mathbb{Z})$: is $x \in G$?

Membership

Given $G \leq \mathrm{GL}(d, \mathbb{Z})$, and $x \in \mathrm{GL}(d, \mathbb{Z})$: is $x \in G$?

Mihailova (1958): membership problem is undecidable for $d \geq 4$.

Membership

Given $G \leq \mathrm{GL}(d, \mathbb{Z})$, and $x \in \mathrm{GL}(d, \mathbb{Z})$: is $x \in G$?

Mihailova (1958): membership problem is undecidable for $d \geq 4$.

$\mathrm{GF}(q)$: $|\mathrm{GL}(d, q)| = O(q^{d^2})$

Membership

Given $G \leq \mathrm{GL}(d, \mathbb{Z})$, and $x \in \mathrm{GL}(d, \mathbb{Z})$: is $x \in G$?

Mihailova (1958): membership problem is undecidable for $d \geq 4$.

$\mathrm{GF}(q)$: $|\mathrm{GL}(d, q)| = O(q^{d^2})$

Membership decidable from exhaustive search.

Membership

Given $G \leq \mathrm{GL}(d, \mathbb{Z})$, and $x \in \mathrm{GL}(d, \mathbb{Z})$: is $x \in G$?

Mihailova (1958): membership problem is undecidable for $d \geq 4$.

$\mathrm{GF}(q)$: $|\mathrm{GL}(d, q)| = O(q^{d^2})$

Membership decidable from exhaustive search.

Even for $\ldots 1 \times 1$ matrices over $\mathrm{GF(q)}$:
membership related to

Membership

Given $G \leq \mathrm{GL}(d, \mathbb{Z})$, and $x \in \mathrm{GL}(d, \mathbb{Z})$: is $x \in G$?

Mihailova (1958): membership problem is undecidable for $d \geq 4$.

$\mathrm{GF}(q)$: $|\mathrm{GL}(d, q)| = O(q^{d^2})$

Membership decidable from exhaustive search.

Even for ... 1×1 matrices over $\mathrm{GF}(\mathrm{q})$:
membership related to

Discrete log problem
$F = \mathrm{GF}(q)$, $\omega \in F$ primitive.
Given $\alpha \in F$, determine k so that $\alpha = \omega^k$.

Membership

Given $G \leq \mathrm{GL}(d, \mathbb{Z})$, and $x \in \mathrm{GL}(d, \mathbb{Z})$: is $x \in G$?

Mihailova (1958): membership problem is undecidable for $d \geq 4$.

$\mathrm{GF}(q)$: $|\mathrm{GL}(d, q)| = O(q^{d^2})$

Membership decidable from exhaustive search.

Even for . . . 1×1 matrices over $\mathrm{GF}(q)$:
membership related to

Discrete log problem
$F = \mathrm{GF}(q)$, $\omega \in F$ primitive.
Given $\alpha \in F$, determine k so that $\alpha = \omega^k$.

No polynomial-time algorithm known.

Challenge Problem I: Order of a matrix

Let $g \in \mathrm{GL}(d, q)$.
Find $n \geq 1$ such that $g^n = 1$.

Challenge Problem I: Order of a matrix

Let $g \in \mathrm{GL}(d, q)$.
Find $n \geq 1$ such that $g^n = 1$.

$\mathrm{GL}(d, q)$ has elements of order $q^d - 1$ (Singer cycles)

Challenge Problem I: Order of a matrix

Let $g \in \mathrm{GL}(d, q)$.
Find $n \geq 1$ such that $g^n = 1$.

$\mathrm{GL}(d, q)$ has elements of order $q^d - 1$ (Singer cycles)

To find $|g|$: probably requires factorisation of numbers of form $q^i - 1$, a hard problem.

Challenge Problem I: Order of a matrix

Let $g \in \mathrm{GL}(d, q)$.
Find $n \geq 1$ such that $g^n = 1$.

$\mathrm{GL}(d, q)$ has elements of order $q^d - 1$ (Singer cycles)

To find $|g|$: probably requires factorisation of numbers of form $q^i - 1$, a hard problem.

Babai & Beals (1999):

Theorem

If the set of primes dividing a multiplicative upper-bound B for $|g|$ is known, then the precise value of $|g|$ can be determined in polynomial time.

Celler & Leedham-Green (1995): compute order in time $O(d^3 \log q)$ subject to factorisation of $q^i - 1$ for $1 \leq i \leq d$.

Celler & Leedham-Green (1995): compute order in time $O(d^3 \log q)$ subject to factorisation of $q^i - 1$ for $1 \leq i \leq d$.

- Compute a "good" multiplicative upper bound E for $|g|$.

Celler & Leedham-Green (1995): compute order in time $O(d^3 \log q)$ subject to factorisation of $q^i - 1$ for $1 \leq i \leq d$.

- Compute a "good" multiplicative upper bound E for $|g|$.

Determine and factorise minimal polynomial for g as

$$m(x) = \prod_{i=1}^{t} f_i(x)^{m_i}$$

where $\deg(f_i) = d_i$ and $\beta = \lceil \log_p \max m_i \rceil$.

Celler & Leedham-Green (1995): compute order in time $O(d^3 \log q)$ subject to factorisation of $q^i - 1$ for $1 \leq i \leq d$.

- Compute a "good" multiplicative upper bound E for $|g|$.

Determine and factorise minimal polynomial for g as

$$m(x) = \prod_{i=1}^{t} f_i(x)^{m_i}$$

where $\deg(f_i) = d_i$ and $\beta = \lceil \log_p \max m_i \rceil$.

$E = \mathrm{lcm}(q^{d_i} - 1) \times p^\beta$

Celler & Leedham-Green (1995): compute order in time $O(d^3 \log q)$ subject to factorisation of $q^i - 1$ for $1 \leq i \leq d$.

- Compute a "good" multiplicative upper bound E for $|g|$.

Determine and factorise minimal polynomial for g as

$$m(x) = \prod_{i=1}^{t} f_i(x)^{m_i}$$

where $\deg(f_i) = d_i$ and $\beta = \lceil \log_p \max m_i \rceil$.

$E = \mathrm{lcm}(q^{d_i} - 1) \times p^\beta$

$|g|$ divides E.

How can we use E?

If $E = \prod_{i=1}^{t} p_i^{\alpha_i}$ then we can determine $|g|$ in $O(\log t \log n)$ multiplications.

How can we use E?

If $E = \prod_{i=1}^{t} p_i^{\alpha_i}$ then we can determine $|g|$ in $O(\log t \log n)$ multiplications.

If $t = 1$, then compute $g^{p_1^j}$ for $j = 1, 2, \ldots, \alpha_1$.

How can we use E?

If $E = \prod_{i=1}^{t} p_i^{\alpha_i}$ then we can determine $|g|$ in $O(\log t \log n)$ multiplications.

If $t = 1$, then compute $g^{p_1^j}$ for $j = 1, 2, \ldots, \alpha_1$.

Otherwise write $E = uv$ where u, v are coprime and have approximately same number of distinct prime factors.

How can we use E?

If $E = \prod_{i=1}^{t} p_i^{\alpha_i}$ then we can determine $|g|$ in $O(\log t \log n)$ multiplications.

If $t = 1$, then compute $g^{p_1^j}$ for $j = 1, 2, \ldots, \alpha_1$.

Otherwise write $E = uv$ where u, v are coprime and have approximately same number of distinct prime factors.

Now g^u has order k say, dividing v;

How can we use E?

If $E = \prod_{i=1}^{t} p_i^{\alpha_i}$ then we can determine $|g|$ in $O(\log t \log n)$ multiplications.

If $t = 1$, then compute $g^{p_1^j}$ for $j = 1, 2, \ldots, \alpha_1$.

Otherwise write $E = uv$ where u, v are coprime and have approximately same number of distinct prime factors.

Now g^u has order k say, dividing v;
and g^k has order ℓ say, dividing u.

How can we use E?

If $E = \prod_{i=1}^{t} p_i^{\alpha_i}$ then we can determine $|g|$ in $O(\log t \log n)$ multiplications.

If $t = 1$, then compute $g^{p_1^j}$ for $j = 1, 2, \ldots, \alpha_1$.

Otherwise write $E = uv$ where u, v are coprime and have approximately same number of distinct prime factors.

Now g^u has order k say, dividing v;
and g^k has order ℓ say, dividing u.

The order of g is $k\ell$.

So cost is $O(d^3 \log q \log t)$ field operations if we can *factorise* E.

So cost is $O(d^3 \log q \log t)$ field operations if we can *factorise* E.

If we don't complete the factorisation, then obtain *pseudo-order* [order $\times$ some large primes] of g

So cost is $O(d^3 \log q \log t)$ field operations if we can *factorise* E.

If we don't complete the factorisation, then obtain *pseudo-order* [order $\times$ some large primes] of g suffices for most theoretical and practical purposes.

So cost is $O(d^3 \log q \log t)$ field operations if we can *factorise* E.

If we don't complete the factorisation, then obtain *pseudo-order* [order $\times$ some large primes] of g suffices for most theoretical and practical purposes.

Implementations in both GAP and Magma use databases of factorisations of numbers of the form $q^i - 1$, prepared as part of the Cunningham Project.

Variation on this theme

Task: Determine if g has *even* order.

Variation on this theme

Task: Determine if g has *even* order.

If we just know E, then we can learn in polynomial time the *exact* power of 2 (or of any specified prime) which divides $|g|$.

Variation on this theme

Task: Determine if g has *even* order.

If we just know E, then we can learn in polynomial time the *exact* power of 2 (or of any specified prime) which divides $|g|$.

By repeated division by 2, we write $E = 2^m b$ where b is odd.

Variation on this theme

Task: Determine if g has *even* order.

If we just know E, then we can learn in polynomial time the *exact* power of 2 (or of any specified prime) which divides $|g|$.

By repeated division by 2, we write $E = 2^m b$ where b is odd.

Now we compute $h = g^b$, and determine (by powering) its order which divides 2^m.

Randomness

$$|\mathrm{GL}(d,q)| = O(q^{d^2})$$

Randomness

$|\mathrm{GL}(d,q)| = O(q^{d^2})$

Many algorithms are **randomised**: use random search in G to find elements having prescribed property $\mathcal{P}$.

Example

- Characteristic polynomial having factor of degree $> d/2$.
- Order divisible by prescribed prime.

Randomness

$|\mathrm{GL}(d, q)| = O(q^{d^2})$

Many algorithms are **randomised**: use random search in G to find elements having prescribed property $\mathcal{P}$.

Example

- Characteristic polynomial having factor of degree $> d/2$.
- Order divisible by prescribed prime.

Common feature: algorithms depend on detailed analysis of **proportion** of elements of finite simple groups satisfying $\mathcal{P}$.

Assume we determine a lower bound, say $1/k$, for proportion of elements in G satisfying Property $\mathcal{P}$.

Assume we determine a lower bound, say $1/k$, for proportion of elements in G satisfying Property $\mathcal{P}$.

To find element satisfying $\mathcal{P}$ by random search with a probability of failure less than given $\epsilon \in (0, 1)$: choose a sample of uniformly distributed random elements in G of size at least $\lceil -\log_e(\epsilon) \rceil k$.

Challenge Problem II: Generate random elements

Challenge Problem II: Generate random elements

Babai (1991): Vertex-transitive graph approach

Independent nearly uniformly random distributed elements of finite group $G = \langle X \rangle$ can be found after a preprocessing stage consisting of $O(\log^5 |G|)$ group operations.

Challenge Problem II: Generate random elements

Babai (1991): Vertex-transitive graph approach
Independent nearly uniformly random distributed elements of finite group $G = \langle X \rangle$ can be found after a preprocessing stage consisting of $O(\log^5 |G|)$ group operations.

Preprocessing proceeds in $O(\log |G|)$ phases.

Challenge Problem II: Generate random elements

Babai (1991): Vertex-transitive graph approach
Independent nearly uniformly random distributed elements of finite group $G = \langle X \rangle$ can be found after a preprocessing stage consisting of $O(\log^5 |G|)$ group operations.

Preprocessing proceeds in $O(\log |G|)$ phases.

In each phase, random walk of random length between 1 and $O((\log |G|)^4)$ performed on Cayley graph of G.

Challenge Problem II: Generate random elements

Babai (1991): Vertex-transitive graph approach
Independent nearly uniformly random distributed elements of finite group $G = \langle X \rangle$ can be found after a preprocessing stage consisting of $O(\log^5 |G|)$ group operations.

Preprocessing proceeds in $O(\log |G|)$ phases.

In each phase, random walk of random length between 1 and $O((\log |G|)^4)$ performed on Cayley graph of G.

Element found when walk finished is added to generators of G.

Challenge Problem II: Generate random elements

Babai (1991): Vertex-transitive graph approach
Independent nearly uniformly random distributed elements of finite group $G = \langle X \rangle$ can be found after a preprocessing stage consisting of $O(\log^5 |G|)$ group operations.

Preprocessing proceeds in $O(\log |G|)$ phases.

In each phase, random walk of random length between 1 and $O((\log |G|)^4)$ performed on Cayley graph of G.

Element found when walk finished is added to generators of G.

Walk is repeated $O(\log |G|)$ times.

Final list S of $O(\log |G|)$ elements input to construction phase.

Final list S of $O(\log|G|)$ elements input to construction phase.

Random element is *random subproduct* of S:

$$g_1^{\epsilon_1} \dots g_m^{\epsilon_m}$$

where $S = \{g_1, \dots, g_m\}$ and $\epsilon_i \in \{0, 1\}$ (chosen independently).

Final list S of $O(\log|G|)$ elements input to construction phase.

Random element is *random subproduct* of S:

$$g_1^{\epsilon_1} \dots g_m^{\epsilon_m}$$

where $S = \{g_1, \dots, g_m\}$ and $\epsilon_i \in \{0, 1\}$ (chosen independently).

For $G \leq \mathrm{GL}(d, q)$, $\log|G| < d^2 \log q$.

Final list S of $O(\log|G|)$ elements input to construction phase.

Random element is *random subproduct* of S:

$$g_1^{\epsilon_1} \dots g_m^{\epsilon_m}$$

where $S = \{g_1, \dots, g_m\}$ and $\epsilon_i \in \{0, 1\}$ (chosen independently).

For $G \leq \mathrm{GL}(d, q)$, $\log|G| < d^2 \log q$.

Initialisation phase $O(d^{10} \log^5 q)$.

Final list S of $O(\log|G|)$ elements input to construction phase.

Random element is *random subproduct* of S:

$$g_1^{\epsilon_1}\cdots g_m^{\epsilon_m}$$

where $S=\{g_1,\ldots,g_m\}$ and $\epsilon_i\in\{0,1\}$ (chosen independently).

For $G\leq \mathrm{GL}(d,q)$, $\log|G|<d^2\log q$.

Initialisation phase $O(d^{10}\log^5 q)$.

Cost per random element is $O(\log|G|)$.

CLMNO (1995): Product replacement algorithm

CLMNO (1995): Product replacement algorithm

Input: ordered list of generators $[g_1, \ldots, g_m]$ for G.

CLMNO (1995): Product replacement algorithm

Input: ordered list of generators $[g_1, \ldots, g_m]$ for G.

Accumulator: r initialised to be identity of G.

CLMNO (1995): Product replacement algorithm

Input: ordered list of generators $[g_1, \ldots, g_m]$ for G.

Accumulator: r initialised to be identity of G.

Basic step:

- Select at random i, j where $1 \leq i, j \leq m$.

CLMNO (1995): Product replacement algorithm

Input: ordered list of generators $[g_1, \ldots, g_m]$ for G.

Accumulator: r initialised to be identity of G.

Basic step:

- Select at random i, j where $1 \leq i, j \leq m$.
- Replace g_i by either $g_i g_j$ or $g_j g_i$.

CLMNO (1995): Product replacement algorithm

Input: ordered list of generators $[g_1, \ldots, g_m]$ for G.

Accumulator: r initialised to be identity of G.

Basic step:

- Select at random i, j where $1 \leq i, j \leq m$.
- Replace g_i by either $g_i g_j$ or $g_j g_i$.
- Multiply r by g_i.

CLMNO (1995): Product replacement algorithm

Input: ordered list of generators $[g_1, \ldots, g_m]$ for G.

Accumulator: r initialised to be identity of G.

Basic step:

- Select at random i, j where $1 \leq i, j \leq m$.
- Replace g_i by either $g_i g_j$ or $g_j g_i$.
- Multiply r by g_i.

Basic step repeated a number, say t, of times.

CLMNO (1995): Product replacement algorithm

Input: ordered list of generators $[g_1, \ldots, g_m]$ for G.

Accumulator: r initialised to be identity of G.

Basic step:

- Select at random i, j where $1 \leq i, j \leq m$.
- Replace g_i by either $g_i g_j$ or $g_j g_i$.
- Multiply r by g_i.

Basic step repeated a number, say t, of times.

Now to obtain random element: execute basic operation once, and return r as random element.

Cost: after initialisation, two matrix multiplications.

Cost: after initialisation, two matrix multiplications.

Markov chain: a discrete random process with a finite number of states and it satisfies the property that the next state depends only on the current state.

Cost: after initialisation, two matrix multiplications.

MARKOV CHAIN: a discrete random process with a finite number of states and it satisfies the property that the next state depends only on the current state.

Aperiodic: all states occur with equal probability.

Cost: after initialisation, two matrix multiplications.

MARKOV CHAIN: a discrete random process with a finite number of states and it satisfies the property that the next state depends only on the current state.

Aperiodic: all states occur with equal probability.

Theorem

Let T be set of all m-tuples of generators of G. Then the algorithm constructs a Markov chain over state space T, and if m is at least twice the size of a minimal generating set of generators for G, this Markov chain is connected and aperiodic.

Cost: after initialisation, two matrix multiplications.

MARKOV CHAIN: a discrete random process with a finite number of states and it satisfies the property that the next state depends only on the current state.

Aperiodic: all states occur with equal probability.

Theorem

Let T be set of all m-tuples of generators of G. Then the algorithm constructs a Markov chain over state space T, and if m is at least twice the size of a minimal generating set of generators for G, this Markov chain is connected and aperiodic.

The random walk approaches a limiting distribution at exponential rate $O((1-\delta)^t)$ where t is number of steps taken.

Mixing time

What can we say about the "mixing time", t?

Mixing time

What can we say about the "mixing time", t?

Variety of statistical tests applied to test outcome of algorithm.
Practical: excellent.

Mixing time

What can we say about the "mixing time", t?

Variety of statistical tests applied to test outcome of algorithm.
Practical: excellent.

- Diaconis & Saloff-Coste (1997, 1998):
  $t = O(\delta^2(G, S) \cdot m)$, where $\delta(G, S)$ is the maximal diameter for the Cayley graph of G wrt generating set S.
  Comparison of two Markov chains on different but related state spaces and combinatorics of random paths.

Mixing time

What can we say about the "mixing time", t?

Variety of statistical tests applied to test outcome of algorithm.
Practical: excellent.

- Diaconis & Saloff-Coste (1997, 1998):
  $t = O(\delta^2(G,S) \cdot m)$, where $\delta(G,S)$ is the maximal diameter for the Cayley graph of G wrt generating set S.
  Comparison of two Markov chains on different but related state spaces and combinatorics of random paths.
- Pak (2001): Mixing time is polynomial. Multi-commodity flow technique.

Mixing time

What can we say about the "mixing time", t?

Variety of statistical tests applied to test outcome of algorithm.
Practical: excellent.

- Diaconis & Saloff-Coste (1997, 1998):
  $t = O(\delta^2(G, S) \cdot m)$, where $\delta(G, S)$ is the maximal diameter for the Cayley graph of G wrt generating set S.
  Comparison of two Markov chains on different but related state spaces and combinatorics of random paths.
- Pak (2001): Mixing time is polynomial. Multi-commodity flow technique.
- Lubotzky & Pak (2002):
  Does the group of automorphisms of a free group of rank > 3 have Kazhdan's property (T)? If so, then "graph of states" is well-behaved, giving excellent mixing time.

Permutation groups

Sims (1970, 1971): base and strong generating set (BSGS).
G acts faithfully on $\Omega = \{1, \ldots, n\}$

$G_\epsilon = \{g \in G \mid \epsilon^g = \epsilon\}$.

Permutation groups

Sims (1970, 1971): base and strong generating set (BSGS).
G acts faithfully on $\Omega = \{1, \ldots, n\}$

$G_\epsilon = \{g \in G \mid \epsilon^g = \epsilon\}$.

Base: sequence of points $B = [\epsilon_1, \epsilon_2, \ldots, \epsilon_k]$ where $G_{\epsilon_1, \epsilon_2, \ldots, \epsilon_k} = 1$.

Permutation groups

Sims (1970, 1971): base and strong generating set (BSGS).
G acts faithfully on $\Omega = \{1, \ldots, n\}$

$G_\epsilon = \{g \in G \mid \epsilon^g = \epsilon\}$.

Base: sequence of points $B = [\epsilon_1, \epsilon_2, \ldots, \epsilon_k]$ where $G_{\epsilon_1, \epsilon_2, \ldots, \epsilon_k} = 1$.

This determines chain of stabilisers

$$G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(k-1)} \geq G^{(k)} = 1,$$

where $G^{(i)} = G_{\epsilon_1, \epsilon_2, \ldots, \epsilon_i}$.
S strong generating set: $G^{(i)} = \left\langle S \cap G^{(i)} \right\rangle$

Example

$G = \langle (1,5,2,6), (1,2)(3,4)(5,6) \rangle$
$B = [1,3]$
$G > G_1 > G_{1,3} = 1$
$S = \{(1,5,2,6), (1,2)(3,4)(5,6), (3,4)\}$

Central task: construct *basic orbits* – orbit B_i of the base point ϵ_{i+1} under $G^{(i)}$.

Central task: construct *basic orbits* – orbit B_i of the base point ϵ_{i+1} under $G^{(i)}$.

$$|G^{(i)} : G^{(i+1)}| = \#B_i$$

Central task: construct *basic orbits* – orbit B_i of the base point ϵ_{i+1} under $G^{(i)}$.

$$|G^{(i)} : G^{(i+1)}| = \#B_i$$

Schreier's Lemma gives generating set for each $G^{(i)}$.

Central task: construct *basic orbits* – orbit B_i of the base point ϵ_{i+1} under $G^{(i)}$.

$$|G^{(i)} : G^{(i+1)}| = \#B_i$$

Schreier's Lemma gives generating set for each $G^{(i)}$.

Base image $B^g = [\epsilon_1^g, \ldots \epsilon_k^g]$ *uniquely* determines g:

Central task: construct *basic orbits* – orbit B_i of the base point ϵ_{i+1} under $G^{(i)}$.

$$|G^{(i)} : G^{(i+1)}| = \#B_i$$

Schreier's Lemma gives generating set for each $G^{(i)}$.

Base image $B^g = [\epsilon_1^g, \ldots \epsilon_k^g]$ *uniquely* determines g:

if $B^g = B^h$ then $B^{gh^{-1}} = B$, so $gh^{-1} = 1$. Hence g can be represented as $|B|$-tuple.

Central task: construct *basic orbits* – orbit B_i of the base point ϵ_{i+1} under $G^{(i)}$.

$$|G^{(i)} : G^{(i+1)}| = \#B_i$$

Schreier's Lemma gives generating set for each $G^{(i)}$.

Base image $B^g = [\epsilon_1^g, \ldots \epsilon_k^g]$ *uniquely* determines g:

if $B^g = B^h$ then $B^{gh^{-1}} = B$, so $gh^{-1} = 1$. Hence g can be represented as $|B|$-tuple.

Variations underpin both theoretical and practical approaches to permutation group algorithms.

G acts faithfully on $V = F^d$: $v \cdot g$, for $v \in V$

G acts faithfully on $V = F^d$: $v \cdot g$, for $v \in V$

Compute BSGS for G, viewed as permutation group on the vectors.

Base points: standard basis vectors for V.

Schreier-Sims for matrix groups

G acts faithfully on $V = F^d$: $v \cdot g$, for $v \in V$

Compute BSGS for G, viewed as permutation group on the vectors.

Base points: standard basis vectors for V.

Central problem: basic orbits B_i large. Usually $|B_1|$ is $|G|$.

Schreier-Sims for matrix groups

G acts faithfully on $V = F^d$: $v \cdot g$, for $v \in V$

Compute BSGS for G, viewed as permutation group on the vectors.

Base points: standard basis vectors for V.

Central problem: basic orbits B_i large. Usually $|B_1|$ is $|G|$.

Butler (1979): action of G on one-dimensional subspaces of V.

Schreier-Sims for matrix groups

G acts faithfully on $V = F^d$: $v \cdot g$, for $v \in V$

Compute BSGS for G, viewed as permutation group on the vectors.

Base points: standard basis vectors for V.

Central problem: basic orbits B_i large. Usually $|B_1|$ is $|G|$.

Butler (1979): action of G on one-dimensional subspaces of V.

Murray & O'Brien (1995): heuristic algorithm to select base points.

G acts faithfully on $V = F^d$: $v \cdot g$, for $v \in V$

Compute BSGS for G, viewed as permutation group on the vectors.

Base points: standard basis vectors for V.

Central problem: basic orbits B_i large. Usually $|B_1|$ is $|G|$.

Butler (1979): action of G on one-dimensional subspaces of V.

Murray & O'Brien (1995): heuristic algorithm to select base points.

Neunhöffer et al. (2000s): use "helper subgroups" to construct large orbits

Critical for success: **index of one stabiliser in its predecessor**.

Critical for success: **index of one stabiliser in its predecessor**.

$|S_n : S_{n-1}| = n$

Critical for success: **index of one stabiliser in its predecessor**.

$|S_n : S_{n-1}| = n$

"Optimal" subgroup chain for $\mathrm{GL}(d, q)$?

$$\mathrm{GL}(d, q) \geq q^{d-1}.\mathrm{GL}(d-1, q) \geq \mathrm{GL}(d-1, q) \geq \dots$$

Leading index: $q^d - 1$.

Critical for success: **index of one stabiliser in its predecessor**.

$|S_n : S_{n-1}| = n$

"Optimal" subgroup chain for $\mathrm{GL}(d, q)$?

$$\mathrm{GL}(d, q) \geq q^{d-1}.\mathrm{GL}(d-1, q) \geq \mathrm{GL}(d-1, q) \geq \dots$$

Leading index: $q^d - 1$.

Example

Largest maximal subgroup $2^{11} : M_{24} \leq J_4$ index 173 067 389.

Aschbacher (1984)

G maximal subgroup of $\mathrm{GL}(d, q)$, let V be underlying vector space

Aschbacher (1984)

G maximal subgroup of $\mathrm{GL}(d, q)$, let V be underlying vector space

- G preserves some natural linear structure associated with the action of G on V, and has normal subgroup related to this structure,

Aschbacher (1984)

G maximal subgroup of $\mathrm{GL}(d,q)$, let V be underlying vector space

- G preserves some natural linear structure associated with the action of G on V, and has normal subgroup related to this structure,
- or G is almost simple modulo scalars: $T \leq G/Z \leq Aut(T)$ where T is simple.

Basic strategy

1. Determine (at least one of) its Aschbacher categories.
2. If $N \lhd G$ exists, recognise N and G/N recursively, ultimately obtaining a composition series for the group.

Basic strategy

1. Determine (at least one of) its Aschbacher categories.
2. If $N \lhd G$ exists, recognise N and G/N recursively, ultimately obtaining a composition series for the group.

7 categories giving normal subgroup

Basic strategy

1. Determine (at least one of) its Aschbacher categories.
2. If $N \lhd G$ exists, recognise N and G/N recursively, ultimately obtaining a composition series for the group.

7 categories giving normal subgroup

Example

G acts imprimitively on V, preserving r blocks, so $V = \oplus_{i=1}^{r} V_i$.

1. Determine (at least one of) its Aschbacher categories.
2. If $N \lhd G$ exists, recognise N and G/N recursively, ultimately obtaining a composition series for the group.

7 categories giving normal subgroup

Example

G acts imprimitively on V, preserving r blocks, so $V = \oplus_{i=1}^{r} V_i$.

Then $\phi : G \to S_r$ where $r|d$ and $N = \ker \phi$.

Basic strategy

1. Determine (at least one of) its Aschbacher categories.
2. If $N \lhd G$ exists, recognise N and G/N recursively, ultimately obtaining a composition series for the group.

7 categories giving normal subgroup

Example

G acts imprimitively on V, preserving r blocks, so $V = \oplus_{i=1}^{r} V_i$.

Then $\phi : G \to S_r$ where $r|d$ and $N = \ker \phi$.

CompositionTree: exploits geometry to produce composition series for G, factors are **leaves** of tree.