The matrix group recognition project; some of the tricks

Charles Leedham-Green, Queen Mary, University of London crlg@maths.qmul.ac.uk

For me, the first moment of enlightenment in the matrix group recognition project came in 1984, with the publication of Aschbacher's celebrated analysis of the maximal subgroups of the classical groups. This theorem, roughly summarised, states that any subgroup of a classical group is either very close to being simple, or it maps homomorphically into a 'better' group than the original classical group, by virtue of preserving some geometrical structure. Typically, this 'better' group will be a classical group of smaller rank, or of the same rank, but defined over a smaller field, or it may be a permutation group of small degree, or perhaps simply a cyclic group. Thus we can hope to analyse the image of the homomorphism, for example by recursion. Another moment of enlightenment was seeing how we could find an acceptably small generating set for the kernel of this homomorphism. Another was the realisation that we could cheaply produce high quality random elements of a group, as much discussed at this conference. The problems of how to make Aschbacher's theorem explicit, and how to deal with the various families of simple groups, have required numerous insights, and more are still required.

It is a matter of very great satisfaction that some twenty six years after the publication of Aschbacher's theorem a working version of the matrix group recognition project is now available in MAGMA.

This has been the result of a great deal of work by many people. We make explicit use of the classification of the finite simple groups, and we use a great deal of information about these groups, some of which has been developed explicitly for our project. This makes it impossible to give a comprehensive list of collaborators. I have worked with Eamonn O'Brien almost from the inception of the project. He has written most of the code, and produced many of the ideas. Derek Holt has made major contributions, both in the earliest days of the project, and more recently, as the specification of the algorithm has been seriously upgraded to produce more useful output, contributing both code and fundamental ideas, and my former student Henrik Bäärnhielm is also working on upgrading and maintaining the overall structure of the code, and Heiko Dietrich is working at Auckland on the classical groups.

It will not have escaped the attention of those attending this conference that these ideas have been re-worked by Ákos Seress and Max Neunhöffer in GAP. Of course they have added valuable ideas of their own, some of which we have adopted; principally the use of 'nice generators'.

While trying to get to sleep, and contemplating what to cover in this talk, I tried counting the number of tricks that we have had to produce to get the code to its current state. Having got to 150, perhaps having counted some more than once, I finally fell asleep. I write 'trick' because most of the ideas, once one has thought of them, are quite simple. So rather than trying to discuss the strategic issues, I shall discuss some of the tricks. Some, such as Trick 1 below, come under the heading of linear algebra, or finite field algorithms. These are supplied by MAGMA; we have not contributed to their development.

Obviously some of our algorithms run faster than others; but we aspire to be able to compute with matrices of degree up to about 240, and with fields of order up to 19^{17}. These limits should not be taken too seriously; experts will be able to construct smaller

examples that will defeat us. However, one should imagine an element of such a group, and consider what one might do with it. This brings me to my first item.

Trick 1. *Given $g \in \mathrm{GL}(d,q)$; find the minimum polynomial of g.*

More precisely, an invertible matrix A is constructed so that $A^{-1}gA$ is a sparse matrix from which the minimal polynomial can be read off. A critical issue is the fact that this can be done by an algorithm with complexity $O(d^3)$ field operations.

Trick 2. *Compute the order of $g \in \mathrm{GL}(d,p^e)$.*

This trick is the combination of three simple subtricks.

1. Calculate and factorise the minimum polynomial $f(x)$ of g; so $f(x) = \prod_i f_i(x)^{a_i}$, where the $f_i(x)$ are distinct irreducible monic polynomials over $\mathrm{GF}(p^e)$. The factorisation of univariate polynomials over a finite field is one of the glories of computer algebra. The power of p that divides the order of g can be computed precisely from the exponents (a_i); this power being p^n, where p^{n-1} is the least power of p such that $p^{n-1} < a_i$ for all i. In fact the order of g is $p^{n-1}r$ where r is prime to p, and divides the least common factor of the integers $q^i - 1$.

2. Let H be any finite group, and let $h \in H$. Calculate the order of h given that this order divides n for some given n, We write this in pseudo-code.

```
Order(h, n);
begin
  if h is the identity return 1.
  if n = p^t for some prime p
    order := 1;
    for i := 1 to t − 2 do
      if h is the identity return order;
      h := h^p;
      order := order × p;
    end for;
    return order;
    factorise n into two proper coprime factors, n = n1 n2;
    h1 := h^{n1};
    h2 := h^{n2};
    return Order(h1, n2) × Order(h2, n1);
  end;
```

The critical point here is the choice of factorisation of n into coprime factors. Since n can, for example, be $p^{ed} - 1$, this factorisation may cause problems. It may be that the factorisation of n is given by the Cunningham project or, the Brent–Montgomery–te Riele table; but, if n is outside these limits, a difficulty may arise. However, if n is factorised as $n = n_1 n_2$, where n_1 is a product of small factors, and n_2 is a product of large factors, then $h_1 = 1$, provided that the order of h is a product of small primes; so difficult factorisations are often avoided. Of course the definition of 'small' here is very generous, such is the power of integer factorisation techniques.

3. It will be seen that subtrick 2 involves many multiplications in H. If the usual algorithm for computing h^n is used, the number of multiplications required can approach

$2\log_2 n$. So if n approximates to p^{ed} the number of multiplications approaches $2de\log_2(p)$. If H is $\mathrm{GL}(d,q)$ this will make the algorithm too slow. However, r in subtrick 1 is the multiplicative order of t in the ring $\mathrm{GF}(p^e)[t]/(\prod_i f_i(t))$, and multiplication in polynomial rings is very fast.

Trick 3. *Compute g^n where $g \in \mathrm{GL}(d,p^e)$.*

The problem here is that n may be of order approaching q^d; and we do not wish to carry out $\log n$ group multiplications. In fact we aim for a complexity that is no worse (up to a small constant) than the cost of a single matrix multiplication.

The solution is as follows. Find a matrix A, as in trick 1, so that $h = A^{-1}gA$ is sparse, and the minimal polynomial $f(x)$ of g can be read off. Now, as with trick 2, compute x^n in $\mathrm{GF}(p^e)[x]/(f(x))$; say $x^n = \sum_{i=0}^{d-1} a_i x^i$. Now $h^n = \sum_{i=0}^{d-1} a_i h^i$. To evaluate this expression requires $d-2$ matrix multiplications; but the multiplications are by a sparse matrix, so the complexity of each multiplication is $O(d^2)$. Finally, $g^n = Ah^nA^{-1}$.

The greatest effort in the project has been expended in dealing with simple groups, or more precisely with almost simple groups; that is to say, we have a group G with $S \le G \le \mathrm{Aut}(S)$. Then given $g \in G$ one may need to decide whether $g \in S$, or more precisely to determine the order of g modulo S. The following very simple trick attempts to answer this question.

Trick 4. *Given X, where $G = \langle X\rangle$ satisfies $S \le G \le \mathrm{Aut}(S)$ for some simple group S, and given $g \in G$, estimate the order of g modulo S.*

Construct random elements $g_1, g_2, \ldots, g_m$ of G', and return the least common multiple l of the orders of the m products $g_i g$.

This could hardly be simpler. The question of how to construct random elements of G has been much discussed in this conference, and variations on this theme give random elements of G'. A more serious issue is to decide whether the algorithm gives the correct answer. Clearly l is a multiple of the required answer, In particular, if $l = 1$ then certainly $g \in G'$. If $l > 1$ then one needs to consider the probability that the correct answer is a proper factor of l. Clearly this probability tends to zero as m tends to infinity. To analyse the rate of convergence one has to consider, among other things, the structure of the outer automorphism group of G; but to determine the probability that l will be greater than 1 when in fact $g \in S$ then, of course, one needs only to consider the distribution of elements of various orders within S.

The expectation that the matrix group project would come to a happy conclusion rose slowly from a base very close to zero over the years. One of the more confident predictions of failure was the following. Suppose that we are given a generating set X for $G = \mathrm{SL}(2,p^e)$, find an element of G of order p as a word (or better as a straight line program) in X. The difficulty arises when p^e is big. The proportion of elements of $\mathrm{SL}(2,p^e)$ of order a multiple of p is approximately $2/p^e$, so a random search is inadequate for large p^e. The expectation was that a random search was the best strategy; but this proved to be false.

To appreciate the solution to the problem one has to understand the discrete logarithm problem. In the form that is needed here, this is as follows. One is given a primitive element

a of a finite field $\mathrm{GF}(p^e)$, and another non-zero element of b, the problem is to solve for n the equation $b = a^n$. The field $\mathrm{GF}(p^e)$, and the element a, are defined by giving the minimum polynomial of a over $\mathrm{GF}(p)$. The element b is given as a polynomial in a. The solution of the discete log problem is a central issue in computer algebra. If p^e is less than 2^{20} then MAGMA operates in terms of Zech logarithms, so that discrete logarithms are determined by lookup. But in general, although no polynomial time algorithm is known, discrete logarithms can be computed rapidly enough for our purposes.

The problem of finding an element of order p needs to be solved for all representations of $\mathrm{SL}(2, p^e)$. It is solved by reducing to the natural representation, and then solving the problem in this case. I shall only discuss the natural representation. The significance of the problem is that, once it has been solved, we can easily write any element of $\mathrm{SL}(2, p^e)$ as a straight line program in X.

Trick 5. *Given a generating set X for $G = \mathrm{SL}(2, p^e)$, find an element of G of order p as a straight line program in X.*

The elements of G of order p are the transvections. To find a transvection it suffices to find two non-commuting elements of G that share an eigenvector. Then their commutator will be a transvection. The first step is to find, by random search, an element h of order $q - 1$. By a change of basis, h is diagonalised, with eigenvalues a and a^{-1}, where a is a primitive element of $\mathrm{GF}(p^e)$. Now take a random element $g \in G$ (as a straight line program on X), The condition that, for some $\lambda \in \mathrm{GF}(p^e)$, gk shares an eigenvector with h, where k has eigenvalues λ and λ^{-1}, gives rise to a quadratic equation in λ. There is an approximately even chance that this equation has a solution in $\mathrm{GF}(p^e)$. If it does not, one tries again with a different choice of g. If the equation does have a solution, then solving this equation for λ, and using discrete logarithms to express λ as a power of a, gives rise to a solution to the problem.

The chairman of the session refused the speaker time to discuss the remaining 145 tricks, so this concludes the account of my lecture. To give even a modest bibliography, and appropriate credit, would double the length of this article; so I refer the reader to the bibliography of 'Constructive recognition of classical groups in odd characteristic' by C.R. Leedham-Green and E.A. O'Brien, Journal of Algebra, 2009, where appropriate references for the above tricks are given.