RECENT DEVELOPMENTS IN COMPUTATIONAL ALGEBRA

A REPORT FROM THE GREEN PERSPECTIVE

Steve Linton

Division of Computer Science, St. Andrews

August 14, 1997

INTRODUCTION

Aims of Talk

- Report on Capabilities of Current Software
- Illustrate Advantages of Integrated Systems
- Focus on use for exploration by group theorists not specialising in computation

Overview

- Case Studies:
  - Constructing HS from the number 23 – Alexander Hulpke
  - Quotients of Grigorchuk's group – Mike Newman
- Quick overview of GAP capabilities
- New and forthcoming developments

CONSTRUCTING HS FROM THE NUMBER 23

Polynomials

- We begin with polynomials. Defining an indeterminate gives us $\mathbb{F}_2[x]$

```
gap> x:=Indeterminate(GF(2));
X(GF(2))
gap> x.name:="x";;
```

- Now we can work with this:

```
gap> f:=x^23-1;
Z(2)^0*(x^23 + 1)
gap> Factors(f);
[ Z(2)^0*(x + 1), Z(2)^0*(x^11 + x^10 + x^6 + x^5 + x^
  4 + x^2 + 1), Z(2)^0*(x^11 + x^9 + x^7 + x^6 + x^
  5 + x + 1) ]
gap> f:=First(Factors(f),i->Degree(i)>1);
Z(2)^0*(x^11 + x^10 + x^6 + x^5 + x^4 + x^2 + 1)
```

A Code

- We next define a code, using the contributed package **GUAVA**

```
gap> RequirePackage("guava");
   ___________                   |
  /           \           /   --+--  Version 1.3
 /      |    | |\\        //|    |
|    _  |    | | \\      // |
|     \ |    | |--\\    //--|     Jasper Cramwinckel
 \     ||    | |   \\  //   |     Erik Roijackers
  \___/  \___/ |    \\//    |     Reinald Baart
     Eric Minkes
gap> cod:=GeneratorPolCode(f,23,GF(2));
a cyclic [23,12,1..7]3 code defined by generator
polynomial over GF(2)
```

Code 2

- This is the binary Golay code, we check a well-known property:

```
gap> IsPerfectCode(cod);
true
```

- Adding a parity bit gives us a code with automorphism group M_{24}

```
gap> ext:=ExtendedCode(cod);
a linear [24,12,8]4 extended code
gap> WeightDistribution(ext);
[ 1, 0, 0, 0, 0, 0, 0, 0, 759, 0, 0, 0, 2576, 0, 0, 0,
  759, 0, 0, 0, 0, 0, 0, 0, 1 ]
```

The Group M_{24}

GUAVA uses an external program to compute automorphism groups quickly

```
gap> m24:=AutomorphismGroup(ext); m24.name:="m24";;
Group( ( 1, 2)( 6,15,24,18)( 7,14,12,19)( 9,17,16,20)
(10,21,23,22)(11,13), ( 2, 3)( 6,10)( 7,24)( 9,17)(11,13)
(12,18)(14,21)(15,23), ( 3, 4)( 7,19)(10,22)(11,13)(12,14)
(15,18)(17,20)(21,23), ( 4, 5)( 6,23,21,16)( 7, 9,20,10)
(11,13)(12,15,17,19)(14,22,18,24), ( 5, 9,11,17)
( 6,19,24,21)( 7,23)( 8,20,13,16)(10,12)(14,15,22,18),
( 5, 8)( 7,23)( 9,16)(10,12)(11,13)(14,22)(17,20)(19,21),
( 6,10,19)( 7,14,18)( 8,11,13)( 9,17,16)(12,21,24)
(15,23,22), ( 6,19)( 7, 9)(10,20)(12,16)(14,24)(15,21)
(17,23)(18,22), ( 6, 7)( 9,19)(10,15)(12,24)(14,16)(17,22)
(18,23)(20,21), ( 6,18)( 7,23)( 9,17)(10,12)(14,21)(15,24)
(16,20)(19,22), ( 6,15)( 7,10)( 9,20)(12,23)(14,22)(16,17)
(18,24)(19,21) )

gap> Size(m24);
244823040
```

The Group M_{22}

$M_{22}.2$ is a two-set stabilizer in M_{24}

```
gap> m22a:=Stabilizer(m24,[23,24],OnSets);;
gap> Size(m22a);
887040
gap> m22a:=Operation(m22a,[1..22]);
Group( ( 1, 3)( 5,17)( 6,18)( 7,10)( 8,16)( 9,13)(11,20)
(21,22), ( 3, 4)( 5,13)( 6,15)( 7,10)( 9,21)(14,16)(17,22)
(19,20), ( 1, 2)( 6,15)( 7,10)( 8,11)( 9,19)(14,17)(16,22)
(20,21), ( 4,16)( 6,21)( 7,19)( 8,12)( 9,20)(10,15)(11,18)
(13,22), ( 1, 2)( 6, 7)( 8,11)( 9,20)(10,15)(12,18)
(19,21) )
```

Towards The Higman-Sims Graph

A Touch of Theory

- We know that HS acts on a graph with $M_{22}.2$ as point stabilizer, and suborbit sizes 1,22 and 77.
- We need an orbit on 77 points. The ATLAS tells us that M_22 has a maximal subgroup:

```
Order   Index  Structure  G.2       Abstract   Mathieu
 5760      77  2^4:A_6    2^4:S_6   N(2A^4)    hexad
```

- So this our action, with point stabilizer $H = 2^4 : S_6$
- H contains a Sylow 2 subgroup of $M_{22}.2$, which must have a normal subgroup $E = 2^4$
- This will be our route to finding H

Finding E

- A Sylow 2 subgroup is polycyclic, so we can use fast methods:

```
gap> s:=SylowSubgroup(m22a,2);;
gap> a:=AgGroup(s);
Group( g1, g2, g3, g4, g5, g6, g7, g8 )
gap> n:=Filtered(NormalSubgroups(a),i->Size(i)=16
>          and IsElementaryAbelian(i));;
gap> n[1];
Subgroup( Group( g1, g2, g3, g4, g5, g6, g7, g8 ),
    [ g1, g3, g6*g7, g8 ] )
```

- Now we change back to permutations:

```
gap> n:=List(n,i->Image(a.bijection,i));;
```

- Finally, we happen to know that E is regular:

```
gap> e:=First(n,i->IsRegular(i,PermGroupOps.MovedPoints(i)));;
```

Making the 77 Point Action

- We make H and then act on its cosets:

```
gap> h:=Normalizer(m22a,e);;
gap> mop:=Operation(m22a,RightCosets(m22a,h),OnRight);;
gap> DegreeOperation(mop,[1..100]);
77
gap> ophom:=OperationHomomorphism(m22a,mop);;
```

- `ophom` is the link between the 22 and 77 point actions

The 100 point action

We combine the 22 and 77 point actions:

```
gap> dp:=DirectProduct(m22a,mop);;
gap> emb1:=Embedding(m22a,dp,1);;
gap> emb2:=Embedding(mop,dp,2);;
gap> diag:=List(m22a.generators,
>            i->Image(emb1,i)*Image(emb2,Image(ophom,i)));;
gap> diag:=Group(diag,());;
gap> diag.name:="M22.2-99";
```

We can simply regard 100 as the fixed point.

Now the Higman-Sims Graph

- We use another contributed package

```
gap> RequirePackage("grape");

Loading  GRAPE 2.31  (GRaph Algorithms using PErmutation groups),
by L.H.Soicher@qmw.ac.uk.
```

- GRAPE works with graphs, making maximum use of known automorphisms. We start with an empty graph on which $M_{22}.2$ acts and adjoin edge orbits

```
gap> gamma:=NullGraph(diag,100);
rec(
  isGraph := true,
  order := 100,
  group := M22.2-diag,
  schreierVector :=
   [ -1, 3, 1, 2, 1, 4, 5, 1, 2, 4, 3, 4, 4, 2, 2, 4, 3,
      4, 5, 1, 1, 3, -2, 5, 4, 1, 5, 5, 4, 2, 4, 4, 3, 4,
      4, 2, 3, 4, 1, 4, 5, 1, 4, 2, 2, 3, 1, 5, 1, 5, 4,
      2, 2, 2, 5, 2, 5, 3, 3, 4, 5, 3, 5, 2, 5, 3, 3, 1,
      3, 4, 1, 3, 1, 1, 1, 4, 1, 4, 3, 5, 5, 5, 3, 3, 3,
      1, 2, 4, 4, 4, 2, 2, 2, 4, 1, 1, 1, 1, 2, -3 ],
  adjacencies := [ [  ], [  ], [  ] ],
  representatives := [ 1, 23, 100 ],
  isSimple := true )
```

Building up the graph 1

- First connect the point 100 to the orbit of size 22

```
gap> AddEdgeOrbit(gamma,[1,100]);AddEdgeOrbit(gamma,[100,1]);
```

- The ATLAS tell us:

> each of the remaining 77 vertices is joined to 6 of these [the 22] points, and may be labelled by the corresponding hexad

- We find the hexad fixed by H and make these edges:

```
gap> hexad:=First(Orbits(h,[1..22]),i->Length(i)=6);
[ 1, 13, 15, 18, 17, 19 ]
gap> AddEdgeOrbit(gamma,[hexad[1],23]);
gap> AddEdgeOrbit(gamma,[23,hexad[1]]);
```

Building up the Graph 2

- There are also edges within the size 77 orbit:

  Two of these 77 vertices are joined just if the corresponding hexads are disjoint.

```
gap> stab := Stabilizer(diag,23);;
gap> orbs := Orbits(stab,[24..99]);;
gap> orbreps := List(orbs,o->o[1]);
[ 24, 43 ]
gap> Intersection(hexad, Adjacency(gamma, 24));
[ 1, 13 ]
gap> Intersection(hexad, Adjacency(gamma, 43));
[ ]
gap> AddEdgeOrbit(gamma,[23,43]); AddEdgeOrbit(gamma,[43,23]);
```

- Now we have made the graph we want

Checking the Graph

- We can look at a few properties:

```
gap> IsSimpleGraph(gamma);
true
gap> Adjacency(gamma,23);
[ 1, 13, 15, 17, 18, 19, 43, 44, 47, 48, 59, 61, 70, 71,
  76, 77, 82, 84, 88, 90, 92, 94 ]
gap> IsDistanceRegular(gamma);
true
```

At Last the Higman-Sims Group

- Finally, we take the automorphism group of `gamma`. Once again an external program is used, but we don't see it:

```
gap> aug:=AutGroupGraph(gamma);;
gap> Size(aug);
88704000
gap> DisplayCompositionSeries(aug);
(G) (3 gens, size 88704000)
 | Z(2)
(S) (2 gens, size 44352000)
 | HS
(1) (0 gens)
```

Summary

- From a polynomial factoring over $\mathbb{F}_2$ we get a code
- From the code, we get M_{24} and hence $M_{22} : 2$
- In $M_{22} : 2$ on 22 points, we find a subgroup of index 77
- We construct a 100 point graph on which $M_{22} : 2$ acts
- The full automorphism group is $HS : 2$

Points to Note

- We have turned a piece of mathematics relatively painlessly into a program
- At every stage, the full structure of all the objects is exposed. We can ask GAP questions about each code, graph or group.
- The whole thing runs in 45 seconds on a Pentium 90 laptop
- We have combined techniques from a number of areas
- GAP provided common interfaces and data structures for them all
- A little extra knowledge, of the problem and of GAP helps

QUOTIENTS OF GRIGORCHUK'S GROUP

The Problem

- A group G_r is defined by an action on a homogeneous tree
- Finite quotients are obtained by considering only the action on the top few levels of the tree
- We wish to study these, and their normal and characteristic subgroups

The Approach

- We write short programs to do the (truncated) construction
- Then we explore the groups interactively
- This hybrid of programming and interaction is a very common way of working with GAP.

Programming the Construction

- G_r is generated by two kinds of elements
- One generator permutes the branches directly below the root
- Others fix a "spine" and permute other subtrees according to a repeating pattern
- We write GAP functions to compute these to a specified level, taking as input the shape of the tree and the patterns of permutations
- Various input checks are omitted for brevity

The Program for the Top level permutation

```
# The element (usually called a) that acts at the root
# nbar is the tree shape, perm the top level permutation
# level is the amount of tree to look at

TopElement := function(nbar, perm, level)
    local blocksize,ims,n1;
    if IsInt(nbar) then
        nbar := List([1..level],x->nbar);
    fi;
    n1 := nbar[1];
    blocksize := Product(nbar{[2..Length(nbar)]});
    ims := Concatenation(List([1..n1],
        i-> [(i^perm - 1)*blocksize + 1 .. i^perm*blocksize]));
    return PermList(ims);
end;
```

The Program for the Other Generators

```
SequenceElement := function(nbar, seq, level)
    local ims,i,blocksize,n,x,perms,offset;
    offset := 0;    ims := [];
    if IsInt(nbar) then
        nbar := List([1..level],i->nbar);
    fi;
    blocksize := Product(nbar{[2..Length(nbar)]});
    for i in [1..level-1] do
        n := nbar[i+1]; blocksize := blocksize/n;
        perms := seq[(i -1) mod Length(seq) +1];
        if not IsList(perms) then perms := [perms]; fi;
        for x in perms do
            Append(ims,Concatenation(List([1..n],
                    i-> offset + [(i^x - 1)*blocksize + 1 .. i^x*blocksize])));
            offset := offset + n*blocksize;
    od; od;
    Add(ims,offset+1);
    return PermList(ims);
end;
```

Programs to Make the Group We Want

- Now we add a few lines to make the generators of the specific group we want

```
GenA := level -> TopElement(2,(1,2),level);
GenB := level -> SequenceElement(2,[(1,2),(1,2),()],level);
GenC := level -> SequenceElement(2,[(1,2),(),(1,2)],level);

MakeGroup := level->Group(GenA(level),GenB(level),GenC(level));
```

- We put these in a file and try them out

```
gap> Read("grigor.g");
gap> MakeGroup(5);
Group( ( 1,17)( 2,18)( 3,19)( 4,20)( 5,21)( 6,22)( 7,23)( 8,24)( 9,25)(10,26)
(11,27)(12,28)(13,29)(14,30)(15,31)(16,32), ( 1, 9)( 2,10)( 3,11)( 4,12)
( 5,13)( 6,14)( 7,15)( 8,16)(17,21)(18,22)(19,23)(20,24)(29,30), ( 1, 9)
( 2,10)( 3,11)( 4,12)( 5,13)( 6,14)( 7,15)( 8,16)(25,27)(26,28)(29,30) )
gap> g := last;; Size(g);
4194304
gap> Collected(Factors(last));
[ [ 2, 22 ] ]
```

Analysing the Group 1

- Since this group actually is soluble, it will be more efficient to treat it as such

```
gap> a := AgGroup(g);
Group( g1, g2, g3, g4, g5, g6, g7, g8, g9, g10, g11, g12,
g13, g14, g15, g16, g17, g18, g19, g20, g21, g22 )
```

- Now we can compute all the normal subgroups – this takes an hour or so

```
gap> ns := NormalSubgroups(a);;
gap> Length(ns);
77
```

Analysing 2

- Now we wish to know which are characteristic
- We use the ANU pq share package to compute generators for the automorphism group of `g`.

```
gap> RequirePackage("anupq");
gap> ag := AutomorphismsPGroup(a);;
gap> Length(ag);  # generators
29
```

- Finally we check which of our 77 normal subgroups are not characteristic

```
gap> Filtered([1..77],i->not ForAll(ag,phi->ForAll(ns[i].generators,
> x->Image(phi,x) in ns[i])));
[  ]
```

- So they all are

Points to Note

- We used basically standard p-group tools for the hard jobs
- But GAP makes it easy to prepare input and analyse results flexibly
- We programmed the fiddly construction but did the analysis interactively
- In GAP 4 we could do the whole calculation with the permutation group, using p-group methods

CONCLUDING REMARKS

What can be done

With a little patience, on a large PC, in GAP or MAGMA:

- Permutation groups up to degree 10^6 for many purposes
- Backtrack searching (centralizer etc.) up to degree 10^5 in many cases
- Conjugacy classes and double cosets are much harder
- Nilpotent groups with $\log |G|$ up to 10^3 for many purposes
- Soluble groups a little smaller
- Computation with words in free generators
- Various techniques for exploring quotients of finitely-presented groups
- Modular representations up to dimension 10^3 (more by indirect routes)
- Character and modular character table constructions
- and more

References

- An Introduction to Computational Group Theory, Akos Seress, Notices of the AMS 44, no 6, June/July 1997, pp 671–679
- An Invitation to Computational Group Theory, Joachim Neubüser, Proceedings of "Groups '93 Galway/St Andrews", LMS Lecture Note Series 212, pp 457–475
- `http://www-gap.dcs.st-and.ac.uk/~gap`

Contributed "Share" Packages

- Increasingly important part of GAP
- User contributed and supported
- Some links to stand-alone programs – **nauty**, **anupq**, vector enumerator, C meataxe
- Some pure GAP – CHEVIE, Specht, **autag**, chrystgap, GRAPE (almost), **grim**, crossed module package, **matrix**
- Package refereeing system
- Full details on the Web pages

Recent Developments (some due shortly)

Groups

- Matrix groups – as discussed last week
- Nearly-linear time permutation groups library
- Partition backtrack methods for many problems
- Hybrid methods for soluble permutation or matrix groups
- Polycyclic quotient algorithm
- Methods for infinite nilpotent groups, based on Hall polynomials and Deep Thought

More Recent Developments

Algebraic Broadening

- Algorithms and implementations for other algebraic structures have been much less sophisticated than those for groups
- New methods and code for semigroups and Lie algebras inspired by group theoretic methods
- These can be used to shed light on groups, or in their own right
- The latest methods in computational representation theory also use deep theory and a broad range of computational components

General Things

- A (somewhat) more careful attitude to randomness – coming from the red perspective
- A general move from stand-alone programs to integrated systems
- A growth in databases providing high-quality access to the results of various classification efforts

Finally

- Reasonably easy-to-use computational tools now exist for investigations in a broad and growing range of algebraic areas
- Last week we saw a few of the topics in computational algebra that are alive and blooming
- I should like to repeat Joachim Neubüser's 1993 "Invitation to Computational Group Theory", rephrased – "come on in, the water's lovely"
- If you do use GAP please tell us, and cite the GAP manual
- If you develop applications, consider making them generally available