Using GAP Effectively
Some Tips and Pitfalls

Steve Linton
CoDiMa School 2016

Philosophy

- GAP has two somewhat contradictory design goals
    - to allow users to pose questions in a way that seems natural to a working mathematician and get answers
    - to allow the expert computational mathematician to implement and apply the most advanced techniques to solve hard problems
- The first is achieved to a limited extent.

```
gap> # to find an element of S_9 which is NOT an involution
gap>Filtered(Elements(SymmetricGroup(9)), x-> x*x <> ())[1];
(7,9,8)
gap> time;
1197
```

- Replace 9 by say 15 and you quickly run out of memory.
- 15! is roughly 1.3×10^{12}.
- This talk is about how to start thinking like an expert.

If you don’t need it don’t store it!

```
gap> n = 9;; Filtered(Elements(SymmetricGroup(n)), x-> x*x <> ())[1];
```

- This computes and stores the full list of elements of S_n
- Then it checks each of them to see if it has order dividing 2 and stores a second list of all of those which don’t
- Finally it returns the first one.
- We can stop looking when we find one. GAP even provides a built in function to do this:

```
gap> n = 9;; First(Elements(SymmetricGroup(n)), x-> x*x <> ());
```

- Stopping things as soon as possible is an important principle. In this case though the real problem is computing and storing all the elements
- Let’s explore some alternatives

Enumerators

- Enumerator returns a list of elements of a domain which may be virtual
  - also EnumeratorSorted — but only if you need it
- For many objects it is quick to construct, but may be slower to access

```
gap> e := Enumerator(SymmetricGroup(99));
<enumerator of perm group>
gap> Length(e);
933262154439441526816992388562667004907159682643816214685929638952175999932299\
156089414639761565182862536979208272237582511852109168640000000000000000000000
gap> e[10^100];
(2,60,99,55,54,65,7,16,18,32,70,15,5,37,43,97,19,31,66,30,90,17,29,85,28,67,
27,62,26,34,52,59)(3,44,73,47,95,45,51,68,50,86,49,83,40,36,81,35,93,12,76,11,
75,10,46,9,96,8,53,42,41,22,78,21,38,20,24,63,23,48,39,56,4,6,58,14,80,13,25,
33)
```

- See also EnumeratorOfCartesianProduct, EnumeratorOfTuples and EnumeratorOf Combinations.

Iterators

- Even an Enumerator can be too heavyweight
  - sometimes you don't need to even number the elements, or know how many there are
- For this GAP has Iterators
  - IsDoneIterator and NextIterator operations

```
gap> n := 9;; i := Iterator(SymmetricGroup(n));;
gap> while not IsDoneIterator(i) do x := NextIterator(i); if x*x = () then break; fi; od;
gap> x;
()
```

- or more concisely, thanks to some built-in magic:

```
gap> n := 9;; for x in SymmetricGroup(n) do if x *x = () then break; fi; od;
gap> x;
()
```

- or even

```
gap> n := 9;; First(SymmetricGroup(n), x->x*x = ());
()
```

Randomness

- Sometimes you can't even make an iterator for your group easily, but you know the elements you want exist and are not too rare

- So make pseudo-random elements of the group until you find one

```
gap> g := SL(10,3);
SL(10,3)
gap> repeat x := PseudoRandom(g); until Order(x) = (3^10-1)/2;
gap> Display(x);
 2 . 2 . 1 . . 2 1 1
 . 1 . 1 2 1 1 2 1 2
 . 1 1 1 1 . . . 1 1
 . . . 1 1 2 1 1 2 1
 . 2 2 . 2 2 1 . . .
 2 . 1 . 2 2 2 2 1 2
 2 2 1 1 . 2 2 2 2 2
 2 1 1 . 2 . 1 . 2 .
 2 1 1 . 2 2 1 1 1 .
 . . 2 1 1 2 2 2 2 1
```

But is searching through all the elements the right thing to do in the first place?

- Element order is a conjugacy invariant
- For many groups there are ways of finding conjugacy class representatives that are faster than listing all elements
  - or they might be already known and stored

```
gap> n := 9;; Representative(First(ConjugacyClasses(SymmetricGroup(n)),
c->Representative(c)^2 <> ()));
(1,2,3)
```

- This is one of the most powerful techniques, especially for non-abelian simple groups and things close to them
- Of course if you are really working in S_n you can simply construct the answer as a permutation

Narrowing the Search

```
gap> First(SymmetricGroup(12), x-> OnTuples([1,2,3,4,5],x) = [1,3,5,7,9] and
Order(x) = 7); time;
(2,3,5,9,4,7,12)
11377
```

- For larger values of 12, this get slow.
    - because it searches lots of elements that fix 2 before it looks at anything that moves 1 to 2
- Use a bit of maths
    - the elements that map [1,2,3,4,5] to [1,3,5,7,9] lie is a coset of a sequence stabilizer

```
gap> g := SymmetricGroup(12);; s := Stabilizer(g,[1,2,3,4,5],OnTuples);;
gap> r := RepresentativeAction(g,[1,2,3,4,5],[1,3,5,7,9],OnTuples);
(2,3,5,9,8,6)(4,7)
gap> First(s,x->Order(x*r) = 7)*r;
(2,3,5,9,4,7,6)
```

General Principles

- Searching for an element in a group
    - Don’t write down the list of elements first
    - Stop when you’ve found it
    - Stop looking at other elements as soon as you know they’re not it
        - order of a matrix can be large and a bit slow to compute
        - if all you care about is whether it is 2, just check “IsOne(x*x) and not IsOne(x)”
- Try and identify a subgroup, or coset or conjugacy class that it lies in
    - remember Sylow subgroups!
    - automorphism group sometimes helps too
- Search only in there

Searching For a Subgroup

- Even worse — quite small groups can have very many subgroups
- Some kinds that are eas(ier) to find
    - Cyclic subgroups (via ConjugacyClasses).
    - NormalSubgroups.
    - Derived, Lower Central etc. series.
    - Sylow subgroups.
    - Maximal subgroups (for some groups).
        - MaximalSubgroups will return all subgroups. You are likely to want ony MaximalSubgroupClassReps.
- Ask yourself if one of these lists might include the one you want, or at least help you on your way

Searching for multiple elements

- Conjecture: $U_3(3)$ cannot be generated by three involutions
    - $|U_3(3)| = 6048$
- So we know some things not to do:
    - list all 216G triples of elements of U3(3) and filter out all the ones that generate the group and consist of involutions
        - use IteratorOfTuples to run through all 216G…
        - use IteratorOfCombinations to run through 36G unordered triples
        - the same, but test for involutions first
            - would take a few hours on my laptops
- find the involutions first (there are just 63 of them) and run over triples
    - takes 22 seconds

```
gap> g := PSU(3,3);
<permutation group of size 6048 with 2 generators>
gap> is := Filtered(g, x->Order(x) = 2);;
gap> Length(is);
63
gap> i := IteratorOfCombinations(is,3);; ct := 0;
0
gap> i := IteratorOfCombinations(is,3); while not IsDoneIterator(i) do
> x := NextIterator(i); if Subgroup(g,x) = g then break; fi; ct := ct+1; od;
<iterator>
#G  FULL  736236/  84320kb live   78877K/   4583mb dead    13995/   1180mb free
time;
ct;
Binogap> time;
21859
gap> ct;
39711
gap> Binomial(63,3);
39711
```

Searching for multiple elements

- We still haven't used conjugacy
- We could choose our first involution to be a conjugacy class rep
    - there is only one conjugacy class of involutions
    - reduce search from Binomial(63,3) to Binomial(62,2)
- But now the second involution can be chosen up to conjugacy in the centraliser of the first one
    - just four cases to consider
        - search is now 4*61 cases
- Of course the third one can be chosen up to conjugacy in the normaliser of the subgroup generated by the first two…..
- If the things you are searching for are not all the same, then the order in which you look at them also matters

Morpheus

- This type of search for sequences of elements that generate something is nicely implemented by Alexander Hulpke in a part of the GAP library called Morpheus

- There are various functions that access morpheus documented in the library under “Searching for Homomorphisms”

- Our example is asking whether U3(3) is a quotient of the free product of three cyclic groups of order 2

```
gap> g:=PSU(3,3);;
gap> F:=FreeGroup(3);;
gap> F:=F/[F.1^2,F.2^2,F.3^2];;
gap> GQuotients(F,g);
[ ]
gap> time;
206
```

Morpheus Ctd

```
gap> F:=FreeGroup(2);;
gap> F:=F/[F.1^2,F.2^6];;
gap> GQuotients(F,g);
[ [ f1, f2 ] -> [ (3,4)(5,8)(6,9)(7,10)(20,29)(21,30)(22,31)(23,32)(24,33)(25,34). . . ,
  [ f1, f2 ] -> [ (1,11)(3,20)(4,29)(5,83)(6,74)(7,65)(8,56)(9,47)(10,38)(14,19) . . . ] ]
```

- So $U_3(3)$ is (2,6) generated in two distinct ways.

- Presented as homomorphisms — easy to recover the generators if you want them

- Other Morpheus functions: AllHomomorphisms, AutomorphismGroup, IsomorphicSubgroups

- A powerful tool for many purposes

Anecdote: Extreme Searching

- Looking for 2,3,7 triples in a a permutation group G so big that main memory could only hold two permutations
    - expecting to have to check millions of cases
        - on a 16MHz CPU shared with the entire university
            - this was a while ago
- Know (from character table) all the conjugacy classes of elements order 2 and 3 — and have representatives
- Fix T order 2 and S order 3 which generate G
    - run through all words W in ST and SST up to some length
    - trace points one by one through $(TW^{-1}SW)^7$
    - as soon as one does not get to the start you can discard that W.
- Searches tens of thousands of cases for the cost of one permutation multiply

Working in the right Group

- Mathematicians are very sloppy
  - they constantly identify isomorphic groups
  - So A_5 “is” PSL(2,5) and SL(2,4) and <a,b | $a^2 = b^3 = (ab)^5 = 1$> and <(1,3,6,2,4), (1,2,3)(4,5,6)>
- but computationally these are different
- choose the right one to work in
- Two tools for moving between them:
  - homomorphisms and straight-line programs

Finitely Presented Groups

- Lots of functionality in GAP for fp groups — mostly to do with identifying unknown ones

- Lots of textbooks that define groups by presentations: $D_{2n} = \langle a,b \mid a^n = (ab)^2 = b^2 = 1 \rangle$

- GAP supports some general group theoretic computation with fp groups that turn out to be finite

- But it's usually the wrong way to do things

Finitely Presented Groups

```
gap> f := FreeGroup("a","b");
<free group on the generators [ a, b ]>
gap> AssignGeneratorVariables(f);
#I  Global variable `a' is already defined and will be overwritten
#I  Global variable `b' is already defined and will be overwritten
#I  Assigned the global variables [ a, b ]
gap> g := f/[a^2,b^3,(a*b)^7, Comm(a,b)^8];
<fp group on the generators [ a, b ]>
gap> Sum(Elements(g), Order);;time;
#G  FULL  680861/  77486kb live   85919K/   4610mb dead    12986/   1172mb free
#G  FULL  703271/  76510kb live   36873K/   2643mb dead    13204/   1171mb free
14236
gap> x := Random(g);
b*a^-1*b^-1*(a^-1*b^-1*a^-1*b)^3*(a^-1*b^-1)^3*a*b^-1*a*b*(a*b*a*b^-1)^2*a*b^-1
```

Using Homomorphisms

```
. . . . . .
gap> g := f/[a^2,b^3,(a*b)^7, Comm(a,b)^8];
<fp group on the generators [ a, b ]>
gap> phi := IsomorphismPermGroup(g);
#G  FULL  678461/  75814kb live   50448K/   3000mb dead    12758/   1170mb free
[ a, b ] -> [ (1,2)(3,5)(4,6)(7,11)(8,12)(9,13)(10,14)(16,20)(17,21)(18,22)(19,23)(25,
    29)(26,27)(28,30)(31,34)(32,35)(33,36)(37,41)(38,42)(39,43)(40,44)(45,50)(46,51)(48,
    52)(49,53)(54,56), (2,3,4)(5,7,8)(6,9,10)(11,15,14)(12,16,17)(13,18,19)(20,24,
    23)(21,25,26)(22,27,28)(29,31,32)(30,33,34)(35,37,38)(36,39,40)(41,45,46)(42,47,
    43)(44,48,49)(50,53,54)(51,55,52) ]
gap> h := ImagesSource(phi);
<permutation group of size 10752 with 2 generators>
gap> Sum(Elements(h), Order);; time;
22
gap> x := Random(h);
(1,14,56,52)(2,21,20,41,54,32,42,8)(3,55,23,11)(4,49,47,30,53,6,24,28)(5,13,31,39,45,40,
26,19)(7,34,29,44)(9,46,27,25)(10,12,33,35,48,37,22,17)(15,43,51,50)(16,36,38,18)
gap> PreImagesRepresentative(phi,x);
(b^-1*a^-1*b*a^-1)^3*b^-1*a^-1*b^2*(a*b*a*b^-1)^3*(a*b)^2*a^-1*b*(a^-1*b^-1)^2*a^-1
```

Other Isomorphism Constructors

- Isomorphism[Special]PcGroup
    - pcgroups are usually the fastest representation for solvable groups
- IsomorphismFpGroup
    - basically only if you want a presentation of your group
- SmallerDegreePermRep
    - heuristic
- GAP will sometimes do this for you
    - see ?NiceMonomorphism or ?NiceObject
    - but it can be better to do it by hand

A Few Homomorphism Operations

- Part of general mapping (relation) machinery
- Source and Range (domain and codomain)
    - given when the morphism is constructed
    - morphism does not need to be total or onto, so they may be bigger than you expect
        - ImagesSource and PreImagesRange may be what you want
- Image specialised to ImageElm and ImagesSet
    - which don’t check that the input is in the source
- PreImagesRepresentative gives just ONE preimage
- InverseGeneralMapping
- CompositionMapping

```
gap> g:=Group((1,2,3,4),(1,2),(5,6,7));;
gap> iso:=IsomorphismPcGroup(g);;
gap> h:=Image(iso);;
gap> z:=Centre(h);;
gap> SetCentre(g,PreImage(iso,z));
gap> cl:=ConjugacyClasses(h);;
gap> ncl:=[];;
gap> for c in cl do
> nc:=ConjugacyClass(g,
> PreImage(iso,Representative(c)));;
> SetSize(nc,Size(c));
> SetStabilizerOfExternalSet(nc,
> PreImage(iso,StabilizerOfExternalSet(c)));
> Add(ncl,nc);
> od;
gap> List(ncl,Size);
[ 1, 1, 6, 8, 3, 1, 6, 8, 3, 6, 6, 8, 3, 6, 6 ]
gap> SetConjugacyClasses(g,ncl);
```

Homorphisms in General

- Even if you can't find an isomorphism to a nicer group, you may be able to find a homomorphism
  - solve your problem in the image first and refine

```
gap> g := Group((1,2),(3,4),(5,6),(7,8),(9,10,11),(11,12,13));
Group([ (1,2), (3,4), (5,6), (7,8), (9,10,11), (11,12,13) ])
gap> Number(g, x-> Order(x) mod 2 = 1); Size(g);
45
960
gap> Orbits(g,MovedPoints(g));
[ [ 1, 2 ], [ 3, 4 ], [ 5, 6 ], [ 7, 8 ], [ 9, 10, 11, 12, 13 ] ]
gap> phi := ActionHomomorphism(g,[1..8]);
<action homomorphism>
gap> h := ImagesSource(phi);
Group([ (1,2), (3,4), (5,6), (7,8) ])
gap> odds := Filtered(h, x->Order(x) mod 2 = 1);;
gap> p := PreImagesSet(phi,odds);;
gap> odds := Filtered(h, x->Order(x) mod 2 = 1);; Length(odds);
1
gap> p := PreImagesSet(phi,odds);;
gap> Length(p); Number(p, x->Order(x) mod 2 = 1);
60
45
```

Not all homomorphisms are equal

- If you just make a GroupHomorphismByImages (by giving images of generators)
    - it can be slow to make because it checks (use GroupHomorphismByImagesNC if you are sure you are right)
        - Image and preimage computation can be slow, or preimages can be “nasty” (long words in FP group)
            - essential because factorisation in terms of generators is not always easy
- ActionHomorphisms are usually good
- So are most things produced by IsomorphismXXXGroup

Random Tips 1

- **Avoid long lists of mutable objects**
    - since the objects in the list might change “under its feet” the list can’t remember
        - whether it’s sorted
        - whether the entries are all from the same family
- so whenever you try and search it or call an operation on it, it has to look at every element
- can become very slow
- lists of immutable objects are much better
- sorted lists of immutable comparable objects can use binary search

Random Tips 2

- There are space and time efficient representations of vectors and matrices over finite fields
    - up to order 256 in the kernel
    - bigger fields in package cvec
- Vectors and matrices are not always in these representations by default
    - among other reasons because deciding whether this vector is “really” over GF(3) or GF(9) requires prescience
- ConvertToVectorRep(v, q) and ConvertToMatrixRep(m,q) convert in place
- cvec has its own functions
- working with large uncompressed vectors or matrices is a bad idea.

Further Reading

- A lot of this talk was taken from Alexander Hulpke's talk "Using GAP", especially section 4
    - You can read the original without my mistakes at http://www.math.colostate.edu/~hulpke/paper/gap4tut.pdf
- A lot of similar ideas are found in my paper "The Art and Science of Computing in Large Groups" (in Bosma & van der Poorten: Computational Algebra and Number Theory, 1995, Springer