CONSTRUCTION OF Co_3. AN EXAMPLE OF THE USE OF AN INTEGRATED SYSTEM FOR COMPUTATIONAL GROUP THEORY

ALEXANDER HULPKE AND STEVE LINTON

1. Introduction

This paper aims to demonstrate, by example, a small sample of the capabilities of the GAP system [S+97] for computational algebra. We specifically focus on the advantages arising from the use of an integrated system such as GAP, which allows the easy combination of techniques from a range of areas, without requiring the user to have a detailed knowledge of the algorithms used.

The sporadic group Co_3 has a faithful permutation representation on 276 points which is unusually small for a group of its size. We want to construct this permutation representation by way of a chain of subgroups of ascending order. In this process we will construct explicitly the sporadic simple groups M_{22} and HS together with associated graphs and codes. Our guide in this is the ATLAS of Finite Simple Groups [CCN+85], which contains a variety of information about the groups of interest, including very terse "constructions" – outlines of settings in which these groups can occur.

We will use GAP (version 3.4, patchlevel 4, including the GRAPE [Soi93] and GUAVA [BCMR] share packages) to realise these constructions. We will see that the integration of many algorithms in one system will permit us to follow the path outlined in theory with concrete constructions.

From a computational stand-point, this is not a very large or difficult computation. It is interesting, however, because it uses a very wide range of techniques, and because it demonstrates one important way in which an integrated system such as GAP can be used.

In giving our example, we give the necessary GAP commands in full, although we sometimes abbreviate the resulting output to save space. The input lines and the full output can be found on the web page

`http://www-gap.dcs.st-and.ac.uk/~ahulpke/paper/bathexample.html`

The first author has been supported by EPSRC Grant GL/L21013.

The complete example session took about 16 minutes on the authors' 200MHz PentiumPro PC running Linux and used about 10MB of GAP workspace.

Some of the calculations rely on random selections. This can make it difficult to reproduce the results. The GAP session presented here results from starting GAP 3.4.4 (which initializes the random number generator to a defined state) and calling exactly the commands listed. If the reader repeats this, she should end up with the same objects. In other circumstances or other versions, input (such as explicit points) that is taken from former output has to be modified accordingly. Though the output of some functions that rely on random methods is abbreviated here (but can be seen in full on the mentioned web page) output relevant for later input is always given.

2. The Construction of $M_{22}.2$

We begin with polynomials. By defining an indeterminate, we obtain $\mathbb{F}_2[x]$. We assign a name to this indeterminate so that it will be displayed in a nice way.

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

Now we work with polynomials. We define a small degree polynomial (this is where the number 23 comes in) and factor it. Over $\mathbb{Z}$ it would split just into two factors, but over $\mathbb{F}_2$ we get further factors. We take one of them.

```
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)
```

We next define a code from this polynomial. For this we use the share package GUAVA [BCMR].

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

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

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

If we extend the code by a parity bit, we get the extended Golay code. This code has automorphism group M_{24}. The weight distribution for this extended code, that GUAVA can compute for us, gives the well known numbers of subsets in the corresponding Steiner system.

```
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 ]
```

GUAVA uses an external program desauto [Leo91] to compute the automorphism group of this code. We check that the size of the obtained group is indeed the size of M_{24} and that it acts quintuply transitive.

```
gap> m24:=AutomorphismGroup(ext); m24.name:="m24";
Group( ( 1, 2)( 6,15,24,18)( 7,14,12,19)( 9,17,16,20)
[...]
gap> Size(m24);
244823040
gap> Transitivity(m24,[1..24]);
5
```

$M_{22}.2$ is the stabilizer of a duad (a set of size 2) in M_{24}. As M_{24} acts quintuply transitive it is unimportant which points we select. (The simple group M_{22} itself could be obtained by computing the pointwise stabilizer, using the GAP operation `OnTuples` instead.) The action of $M_{22}.2$ on the remaining points is faithful.

```
gap> m22a:=Stabilizer(m24,[23,24],OnSets);
Subgroup( m24, [ ( 1, 3)( 5,17)( 6,18)( 7,10)( 8,16)
[...]
gap> Size(m22a);
887040
gap> m22a:=Operation(m22a,[1..22]);m22a.name:="m22a";;
Group( ( 1, 3)( 5,17)( 6,18)( 7,10)( 8,16)( 9,13)(11,20)
[...]
gap> Size(m22a);
887040
```

3. The Construction of HS

Now we need a little bit of theory. The ATLAS [CCN+85] tells us on page 80:

*Graph G.2: the automorphism group of the graph of D.G.Higman and C.C. Sims, a rank 3 graph of valence 22 on 100 vertices. Any given vertex has 22 neighbours (*points*), and each of the remaining 77 vertices is joined to 6 of these points and may be labelled by the corresponding hexad. Two of these 77 vertices are joined just if the corresponding hexads are disjoint.*

and on the bottom of the page:

```
Order   Index Structure  G.2    Character   Graph
443520    100 M22      : M22:2  1a+22a+77a  point
```

We deduce that (1) $HS.2$ acts on a graph Γ (the Higman-Sims graph) with 100 vertices (2). There only is one permutation representation of degree 100 for $HS.2$ (6), the point stabilizer is of type $M_{22}.2$. The graph has rank 3 (2), that is the point stabilizer has 3 orbits on the 100 points. A vertex in Γ has 22 neighbours (2) and therefore the point stabilizer must have an orbit of length 22. As it only has 3 orbits there is one remaining orbit of length $100 - 1 - 22 = 77$.

We will construct this graph using the point stabilizer and therefore have to construct $M_{22}.2$ in its intransitive permutation action on 100 points with orbits of lengths 1,22 and 77.

3.1. **A representation of $M_{22}.2$ on 77 points.** The first step of this construction will be to obtain the transitive action of $M_{22}.2$ on 77 points. We will get it by acting on the cosets of a suitable subgroup. To find this subgroup we use the classification of maximal subgroups of M_{22} given in the ATLAS, in which we find the line:

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

This tells us:

- M_{22} has a permutation representation of degree 77 which extends to $M_{22}.2$.
- The point stabilizer in this action is a subgroup $H < M_{22}.2$ of type $2^4 : S_6$. This subgroup has a normal subgroup E of type 2^4, of which it must be the normalizer in $M_{22}.2$.
- As it has odd index 77, H must contain a full Sylow 2 subgroup of $M_{22}.2$. Therefore E, which certainly is contained in a Sylow subgroup, must be normal also in the Sylow 2 subgroup.
- As stabilizer of a hexad, H (and therefore E as well) stabilizes a set of 6 points in the permutation representation on 22 points.

To get H we will look for normal subgroups of order 2^4 in a Sylow 2 subgroup. As a Sylow subgroup is polycyclic, we convert it to an AgGroup. This is a special representation for solvable groups in which the computations we want to do run more quickly.

```
gap> s:=SylowSubgroup(m22a,2);;
gap> a:=AgGroup(s);
Group( g1, g2, g3, g4, g5, g6, g7, g8 )
```

We compute all elementary abelian normal subgroups of size 16 in this group, there are 5 of them.

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

The component `bijection` of the group `a` is an homomorphism back into the permutation group. We use it to get these subgroups as permutation groups.

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

To pick the right E, we could now look at the sizes of the normalizers of the subgroups in `n` and it would turn out that only one of them has the right normalizer size. We happen to know, however, that E acts regularly. (The remaining 6 points form the hexad stabilized by the normalizer.) There only is one group satisfying these conditions. We take this group and compute its normalizer.

```
gap> e:=Filtered(n,i->IsRegular(i,PermGroupOps.MovedPoints(i)));;
gap> Length(e);
1
gap> e:=e[1];;
gap> h:=Normalizer(m22a,e);;
```

Now we compute the action on the cosets.

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

It is worth mentioning, that the following command

```
mop:=Operation(m22a,CanonicalRightTransversal(m22a,h),
  OnCanonicalCosetElements(m22a,h));;
```

produces essentially the same result, but usually works much more quickly and uses less space.

The corresponding operation homomorphism is the link between the two permutation representations.

```
gap> ophom:=OperationHomomorphism(m22a,mop);;
```

3.2. **Representing $M_{22}.2$ on 100 points.** To obtain the action on 100 points we form the direct product of two copies of $M_{22}.2$ acting on 22 and on 77 points respectively and take its diagonal subgroup. We can simply regard 100 as the fixed point.

We get the diagonal subgroup by taking the product of the images of each generator under both embeddings.

```
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)));;
```

We create the group generated by these diagonal elements and give it a name. (Note that GAP requires us to give the identity here, as `diag` is a list which could be empty.)

```
gap> diag:=Group(diag,());;
gap> diag.name:="M22.2-99";
```

3.3. **The Higman-Sims Graph.** It is now time to construct the graph. For this we use the share package GRAPE.

```
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 100 vertices on which $M_{22}.2$ acts and adjoin orbits of edges.

```
gap> gamma:=NullGraph(diag,100);
rec( isGraph := true,
     group := M22.2-99,
     representatives := [ 1, 23, 100 ],
[...]
```

Graphs in GRAPE are directed, so we have to add each edge in two directions. The first edge is to connect the point 100 to the orbit of size 22 to make up for valence 22 (2):

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

Now we connect vertices in the orbit of length 77 with vertices in the orbit of length 22, using line (4) of the description from the ATLAS.

The construction of direct products for permutation groups in GAP maps the first factor on the same permutations and the second factor on permutations shifted by the degree of the first factor. Therefore we can use the representative 23 for the orbit of length 77 and know that this point is the point stabilized by the subgroup $H \leq M_{22}.2$. The corresponding hexad (in the first 22 points) must be stabilized by H.

We have already observed that H has only one orbit of length 6 in its action on 22 points. We fetch this orbit to obtain the hexad and add the corresponding edges. As the computation of the Sylow subgroup used random methods the reader may get a different hexad as a result. See the comment on random methods at the end of the first section.

```
gap> hexad:=First(Orbits(h,[1..22]),i->Length(i)=6);
[ 2, 8, 17, 15, 22, 6 ]
gap> for i in hexad do AddEdgeOrbit(gamma,[i,23]);
> AddEdgeOrbit(gamma,[23,i]); od;
```

Looking at the neighbourhood of 23 we see that we have not yet reached valence 22.

```
gap> Adjacency(gamma,23);
[ 2, 6, 8, 15, 17, 22 ]
```

Indeed, we still have to consider the next rule (5), to determine edges within the orbit of size 77. Because we have a group acting, it is sufficient to test for each representative of the orbits of the stabilizer of 23, whether it is joined to 23. The other edges in the orbits will be added automatically by GRAPE.

```
gap> stab:=Stabilizer(diag,23);;
gap> orbs:=Orbits(stab,[24..99]);;
gap> orbreps:=List(orbs,i->i[1]);
[ 24, 39 ]
```

The hexad corresponding to a point a in the 77-point orbit consists (we constructed the graph that way) of the points in the first orbit adjacent to a. The hexad corresponding to 23 is already known as `hexad`.

```
gap> Intersection(hexad,Adjacency(gamma,24));
[ 15, 17 ]
gap> Intersection(hexad,Adjacency(gamma,39));
[ ]
```

This tells us that 23 must be joined only to 39. We add this edge orbit.

```
gap> AddEdgeOrbit(gamma,[23,39]); AddEdgeOrbit(gamma,[39,23]);
```

Now we have completed the graph we want. We check that it is indeed a simple graph, look at the adjacency of 23 and find out that it is distance regular.

```
gap> IsSimpleGraph(gamma);
true
gap> Adjacency(gamma,23);
[ 2, 6, 8, 15, 17, 22, 39, 42, 46, 49, 52, 56, 63, 68, 70, 76, 80,
  81, 86, 90, 93, 94 ]
```

```
gap> IsDistanceRegular(gamma);
true
```

Finally, we compute the automorphism group of `gamma`. Once again an external program (nauty, [McK90]) is used, but we do not see it:

```
gap> aug:=AutGroupGraph(gamma);;
gap> Size(aug);
88704000
```

Indeed the size is correct for $HS.2$. We can confirm this by looking at the composition series. GAP identifies simple composition factors by their size and the degree of a primitive permutation action, using the classification of finite simple groups.

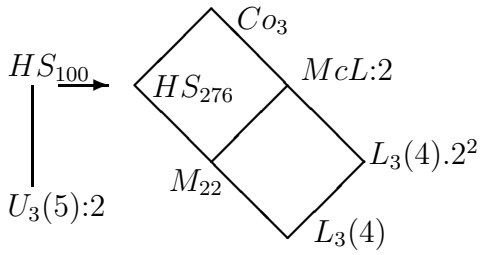
```
gap> DisplayCompositionSeries(aug);
(G) (3 gens, size 88704000)
 | Z(2)
(S) (2 gens, size 44352000)
 | HS
(1) (0 gens)
```

We get HS as the derived subgroup of $HS.2$.

```
gap> hs:=DerivedSubgroup(aug);;
```

4. Getting Co_3

Now we want to construct Co_3. The ATLAS tells us (page 134) that Co_3 has a permutation representation on 276 points in which the point stabilizer is $McL{:}2$. Furthermore HS is a maximal subgroup of Co_3. If we can construct HS in this permutation representation on 276 points, then, together with one further suitably constructed element it will generate Co_3. The construction is not mentioned explicitly in the ATLAS but this "amalgamation" of subgroups is a standard technique for the construction of sporadic groups [PW90].



The picture will illustrate the construction process. The first step will be to construct HS as a permutation group acting on 276 points. The smallest degrees of faithful permutation representations of HS are 100, 176 and 1100. Therefore, as a subgroup of Co_3, HS has two orbits of lengths 100 and 176 respectively. (HS cannot have any fixed points because it is not contained in $McL{:}2$.)

The stabilizer in HS of a point in an orbit of length 176 is of type $U_3(5){:}2$, but there is no description of how to find such a subgroup $U \leq HS$ of index 176 when only the representation of HS on 100

points is known. Instead we will try to find elements by random search in the permutation representation of HS on 100 points that generate such a subgroup U. As the index $[HS : U]$ is relatively small we have good chances. We just have to decide which of the element orders occurring in $U_3(5)$:2 to try to find a suitable U.

To determine our chances of success we use the character tables of $U_3(5)$, $U_3(5)$:2 and HS. We can look them up in the ATLAS but they are more conveniently obtained from a GAP data base.

```
gap> ct:=CharTable("U3(5)");;ct2:=CharTable("U3(5).2");;
gap> cths:=CharTable("hs");;
gap> ct.orders;ct2.orders;cths.orders;
[ 1, 2, 3, 4, 5, 5, 5, 5, 6, 7, 7, 8, 8, 10 ]
[ 1, 2, 3, 4, 5, 5, 5, 6, 7, 8, 10, 2, 4, 6, 8, 10, 12, 20, 20 ]
[ 1, 2, 2, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7, 8, 8, 8, 10, 10,
11, 11, 12, 15, 20, 20 ]
```

The character tables show that $U_3(5)$:2 contains elements of order 12 which $U_3(5)$ does not. Thus an element of order 12 will ensure that we find U and not its derived subgroup U'. Furthermore HS contains only one class of elements of order 12 and this class is self-centralizing. Therefore there is a chance of $1/12$ that a random element of HS has order 12 and we know that there is a conjugate of $U_3(5)$:2 in HS that contains this element. We search for such an element of order 12 which we denote by `e1` and compute its 6th power `e2`.

```
gap> repeat e1:=Random(hs);until OrderPerm(e1)=12;
gap> e2:=e1^6;;
```

As 6 is divisible by 2, `e2` must lie in $U_3(5)$ and is therefore in the first class of elements of order 2. This class in $U_3(5)$:2 has 525 elements

```
gap> ct2.classes[2];
525
```

Along the lines of [Lin95] we can now estimate how much random search is required to find a subgroup of HS isomorphic to $U_3(5) : 2$. In HS, the 6th power of an element of order 12 lies in the second class. (GAP stores only prime power maps from which all power maps can be deduced. Therefore we compute the sixth power as the cube of the square.) We can read from the table that the class of `e2` in HS is of order 5775.

```
gap> cths.orders[21];
12
gap> cths.powermap[3][cths.powermap[2][21]];
2
gap> cths.classes[2];
5775
```

If we take random conjugates of `e2` about every 10th conjugate will lie in a maximal subgroup of type $U_3(5)$:2 which contains also `e1`.

A similar count of pairs for the maximal subgroups of $U_3(5)$:2 (which is left to the reader) establishes that at least every second conjugate of `e2` in $U_3(5)$:2 together with `e1` generates the full group $U_3(5)$:2. Therefore we can assume that in the mean at least one of 20 conjugates of `e2` in HS together with `e1` will generate a maximal subgroup U isomorphic to $U_3(5)$:2 and we can find it by a random search. (As U operates transitively on 100 points the `until` statement has to use the index and cannot check for orbit lengths which otherwise would be quicker.)

```
gap> cnt:=0;;repeat u:=Subgroup(aug,[e1,e2^Random(hs)]);
> cnt:=cnt+1;until Index(hs,u)=176;cnt;
26
```

We were a little bit unlucky in that we had to try 26 random subgroups. (Anyhow this random search took only about 7 seconds on the authors' system.)

We now construct the operation of HS on the cosets of U. Again we use the special operation on canonical coset representatives (which saves over 40MB of memory and several minutes of runtime). We test that its image is indeed primitive. (So U is indeed a maximal subgroup and we don't have to rely on the list of maximal subgroups being complete.) Afterwards we construct a diagonal action of HS on 100+176 points by mirroring the process used above for M_{22}.2.

```
gap> hsop:=Operation(hs,CanonicalRightTransversal(hs,u),
> OnCanonicalCosetElements(hs,u));;
gap> IsPrimitive(hsop,[1..176]);
true
gap> ophom:=OperationHomomorphism(hs,hsop);;
gap> dp:=DirectProduct(hs,hsop);;
gap> emb1:=Embedding(hs,dp,1);;
gap> emb2:=Embedding(hsop,dp,2);;
gap> diag:=List(hs.generators,
>               i->Image(emb1,i)*Image(emb2,Image(ophom,i)));;
gap> diag:=Group(diag,());;diag.name:="hs-276";;
```

4.1. **Search for a further element.** The strategy for finding a further element is as follows: Such elements exist in McL:2. At this point, however, we don't have McL:2 in an appropriate action. We may choose conjugates such that $HS \cap McL{:}2 = M_{22}$ (McL:2 is a stabilizer of a point in the action of Co_3 on 276 points and M_{22} is a point stabilizer in the orbit of length 100 of HS).

Assume that T is a subgroup of HS such that $N_{McL:2}(T) \not\leq HS$. In this case there are elements in $N_{McL:2}(T)$ which are not in HS and we can take such an element as a further generator for Co_3. We will see that (with appropriate choice of T) such elements can be found directly from the automorphism action they induce on T. We shall obtain this action from the automorphism group of T.

A look at the ATLAS list of maximal subgroups of $McL{:}2$ shows that the outer automorphism of M_{22} cannot be realized in $McL{:}2$ and therefore we cannot simply choose $T = M_{22}$. If we look at the point stabilizer in the action of M_{22} on 22 points, we have more luck: This group is of type $L_3(4)$ and while HS contains a maximal subgroup of type $L_3(4).2_1$, $McL{:}2$ contains $L_3(4).2^2$. So we can choose T to be this subgroup $L_3(4).2_1$.

Furthermore, T is just the stabilizer in HS of an edge of the Higman-Sims graph and is therefore easily obtainable. Again we use that the direct product construction mapped the copy of HS acting on 100 points to the same permutations, so we can simply use the graph we have constructed already and do not need to do any point translation.

```
gap> adj:=Adjacency(gamma,1);
[ 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99, 100 ]
gap> t:=Stabilizer(diag,[1,adj[1]],OnSets);;
```

Because we computed T as stabilizer it might have many generators. For the computation of the automorphism group of T, however, it will be helpful to have a subgroup with fewer generators. Therefore we look at subgroups generated by two random elements of T and find quickly two elements which generate a subgroup that equals T and use this generated subgroup as T further on.

```
gap> cnt:=0;;repeat s:=Subgroup(diag,[Random(t),Random(t)]);
> cnt:=cnt+1;until Size(s)=Size(t);cnt;t:=s;
1
```

We know (well, have strong indications. For a proof we would have to classify subgroups of McL isomorphic to $L_3(4)$) that there is an element $g \in McL{:}2$ which normalizes T and is not contained in HS. We want to obtain this element from its automorphism action on T.

4.2. **Finding suitable automorphisms.** Now we start by computing the automorphism group of T. (In principle it should be possible to use the command `AutomorphismGroup` here. There is, however, a problem with GUAVA that we used before: it overwrites the dispatcher function for `AutomorphismGroup`).

```
gap> aus:=t.operations.AutomorphismGroup(t);;
```

```
gap> Size(aus);
241920
```

As we want an outer automorphism, we take the subgroup of inner automorphisms of T (GAP has already computed a generating set and stored it) and consider a transversal in the outer automorphism group. As we are looking for an automorphism of order two, we select those representatives from this transversal whose order modulo the inner automorphisms is 2. (It might be worth mentioning that these few calculations in the automorphism group, together with the calculation of the automorphism group itself, actually take up more than 65 percent of the total runtime!)

```
gap> inner:=Subgroup(aus,aus.innerAutomorphisms);;
gap> Index(aus,inner);
6
gap> rt:=RightTransversal(aus,inner);;
gap> automs:=Filtered(rt,i->i^2 in inner and not i in inner);;
gap> Length(automs);
3
gap> List(automs,i->Order(aus,i));
[ 2, 2, 2 ]
```

We were lucky in that all the coset representatives we got have order 2 (and that is a proof that every coset contains a representative of order 2. The reader starting with different initial data may end up with representatives of composite order. If this is the case, she can multiply these with inner automorphisms to obtain representatives of order 2 for the same cosets. This may require some random searching).

There are three possible representatives. We will continue with all three of them for now. It will turn out later that only one of them is a suitable candidate.

If we look at the orbits of T

```
gap> ot:=Orbits(t,[1..276]);;
gap> List(ot,Length);
[ 2, 56, 42, 56, 120 ]
```

we see that the only orbits of T that its normalizer in Co_3 may swap are those of length 56, the other orbits must remain fixed. (On the other hand, as this normalizer together with HS generates Co_3 it then must swap these to obtain a transitive action.) For each orbit of length $\neq 56$ and for the fused orbit of length 112 we then construct a permutation in the symmetric group on this orbit which by conjugation induces the same action on the image of T projected on this orbit as the automorphism does. Except for the orbit of length 2 this permutation on each

(fused) orbit is unique, as the image of the action of T on the orbit has trivial centralizer in the full symmetric group.

4.3. **Constructing a permutation.** We shall need a small GAP function to compute such a permutation. For this we use a theorem (see for example [DM96, Theorem 4.2B]) which states that an automorphism of a permutation group is induced by conjugation with a permutation in the symmetric group if and only if it maps a point stabilizer to another point stabilizer. In this case, the automorphism maps cosets of the one stabilizer to cosets of the other. This can be used to define a permutation, which then induces this automorphism.

The GAP function `MappingPermListList` creates a permutation which maps one list onto another. From this we will build another function that will try to realize automorphisms on orbits. This function gets a permutation group `grp`, an automorphism `aut` of `grp` and a domain `dom` on which a permutation action will be computed. If the group has two orbits on this domain, the function will create a permutation that will swap both orbits. If the action cannot be induced by a permutation, `false` is returned.

```
PermutationByAutomorphism := function(grp,aut,dom)
local op,oh,p,s,sim,simp,rt,rtim,extelm,l1,l2;
 # We compute the action on the given domain and transfer
 # the automorphism to this permutation action
 op:=Operation(grp,dom);
 oh:=OperationHomomorphism(grp,op);
 aut:=GroupHomomorphismByImages(op,op,op.generators,List(op.generators,
        i->Image(oh,Image(aut,PreImagesRepresentative(oh,i)))));
 aut.isMapping:=true; # just to save time (otherwise GAP will test that
 # it is indeed a homomorphism, but we know this already)

 # compute stabilizer and images
 s:=Stabilizer(op,1);
 sim:=Image(aut,s);

 # is the image a stabilizer? It is if it has an orbit of length 1
 simp:=Filtered(Orbits(sim,[1..Length(dom)]),i->Length(i)=1);
 if Length(simp)=0 then
  return false; # it cannot be induced by a permutation action
 fi;

 # the permutation can be obtained by the induced action on the right
 # cosets.
 simp:=simp[1][1]; #image base point
 rt:=RightTransversal(op,s);
```

```
 rtim:=List(rt,i->Image(aut,i));
 l1:=List(rt,i->1^i);l2:=List(rtim,i->simp^i);

 # we got the images, make a permutation from them.
 if Length(Orbits(grp,dom))=1 then
  extelm:=MappingPermListList(l1,l2);
 else
  # if we have two orbits, we have to ensure they get swapped
  extelm:=MappingPermListList(Concatenation(l1,l2),
                              Concatenation(l2,l1));
 fi;
 # test whether the computed element indeed fulfills the specifictions
 # (This is a safety test only)
 if ForAny(op.generators,i->i^extelm<>i^aut) then
  Error("something went wrong");
 fi;
 # finally move the points acted on to the original domain.
 return extelm^MappingPermListList([1..Length(dom)],dom);
end;
```

It turns out, that of the three candidates we had, only one is induced by a permutation on the orbit of length 120.

```
gap> lo:=First(ot,i->Length(i)=120);;
gap> automs:=Filtered(automs,
>            i->PermutationByAutomorphism(t,i,lo)<>false);;
gap> Length(automs);
1
gap> autom:=automs[1];;
```

Now we simply build a permutation using our function. We get unique permutations for all orbits of length $\notin \{2, 56\}$, and one permutation to swap the two orbits of length 56.

The product of all these is the permutation we need.

```
gap> pos:=Filtered([1..Length(ot)],i->Length(ot[i])=56);
[ 2, 4 ]
gap> perms:=List(ot{Difference([1..5],pos)},
>             i->PermutationByAutomorphism(t,autom,i));;
gap> element:=Product(perms)*PermutationByAutomorphism(t,
>                            autom,Concatenation(ot{pos}));
```

The permutation `element` is unique but for the orbit of length 2. The element we are looking for might swap the two points in this orbit or it might not and so we will have to try both possibilities. In this situation, however, it turns out that swapping both points gives the right permutation and we only show this part of the calculation, for which we change the permutation.

```
gap> ot[1];
[ 1, 79 ]
gap> 1^element;
1
gap> element:=element*(1,79);;
```

Now we can generate Co_3. Again we check that everything went well

```
gap> co3:=Group(Concatenation(diag.generators,[element]),());;
gap> Size(co3);
495766656000
gap> DisplayCompositionSeries(co3);
<G> (3 gens, size 495766656000)
 | Co(3)
<1> (0 gens)
```

Using the same techniques we could now continue and form Co_1 on 98280 points by adding an outer automorphism to $HS < Co_3$. The large number of points however makes this rather elaborate and unsuitable for a presentation in this form.

5. Concluding Remarks

5.1. **Summary of Example and Points to Note.** We started our calculation with a polynomial over $\mathbb{F}_2$ and obtained a code from its factors. From the code we constructed M_{24} and hence $M_{22}.2$. In $M_{22}.2$ acting on 22 points, we found a subgroup of index 77 and so obtained an action on 100 points. We constructed a graph on 100 vertices, on which $M_{22}.2$ acts. The full automorphism group of this graph is $HS.2$. We formed its derived subgroup HS as a subgroup of Co_3 on 276 points and, within HS, a subgroup $L_3(4).2$. From an automorphism of this subgroup we obtained a permutation contained in Co_3, but not in HS. Together with generators of HS, this permutation generates Co_3.

While these constructions were known before in theory, and had been performed relatively explicitly by hand, most of the objects involved are not given explicitly in a book or article. Using the very terse descriptions in the ATLAS, informed by a general knowledge of what computations are likely to be feasible, we have turned a piece of mathematics relatively painlessly into a program. In this presentation, we have, of course, omitted some false starts. Nevertheless, the interactive nature of GAP makes it relatively easy to develop programs of this kind. At every stage, the full structure of all the objects is exposed, so that we can ask GAP mathematical questions about each code, graph or group that we construct.

As well as its considerable library of built-in algorithms, especially for group theory, a key role of GAP in this work was to provide common

interfaces and data structures for techniques from a number of mathematical areas, so that we could, for example, construct M_{24} using tools from coding theory, manipulate it as a group and apply the results in graph theory.

Another point to note is the pattern of working. The combination of interactive capabilities and programmability of GAP allows a middle road between purely interactive exploration and programming. Purely interactive work would not (for example) have allowed us to find the "extra" element of Co_3, because the conversion from automorphisms to permutations was not built into GAP. On the other hand, a non-interactive "design – program – test" model would have been too cumbersome for the main thread of the construction, which is more easily found by interactive experiment. A normal way of working with GAP is to explore interactively, keeping a log file as a record, and stopping, from time to time to program and debug functions that seem necessary or useful, which thereafter act as library extensions.

5.2. **Further Information.** Further information about Computational Group Theory in general can be found in excellent survey articles [Neu95, Ser97]. The GAP system itself, and a great deal of associated information, can be found on the GAP World Wide Web site (`http://www-gap.dcs.st-and.ac.uk/~gap`). In particular the site includes the GAP distribution itself, details about the authors of GAP and of the contributed Share Packages and information about the forthcoming version 4 of the system.

The development of GAP has been supported by several grants (listed under `http://www-gap.dcs.st-and.ac.uk/~gap/Info/funding.html`) for which we are indebted to the granting bodies. We would also like to thank the referee for many helpful comments.

References

[BCMR] Reinalt Baart, Jasper Cramwinckel, Eric Minkes, and Erik Roijackers, *GUAVA: Users manual*.

[CCN+85] J[ohn] H. Conway, R[obert] T. Curtis, S[imon] P. Norton, R[ichard] A. Parker, and R[obert] A. Wilson, ATLAS *of finite groups*, Oxford University Press, 1985.

[DM96] John D. Dixon and Brian Mortimer, *Permutation groups*, Graduate Texts in Mathematics, vol. 163, Springer, Heidelberg, 1996.

[Leo91] Jeffrey S. Leon, *Partition backtrack programs, users manual*, University of Illinois at Chicago, Mathematics Dept., m/c 249, 1991.

[Lin95] S[teve] A. Linton, *The art and science of computing in large groups*, Proceedings of CANT '92 (Wieb Bosma and Alf J. van der Poorten, eds.), Kluwer, 1995, pp. 91–109.

[McK90] B. D. McKay, *nauty user's guide (version 1.5), technical report tr-cs-90-02*, Australian National University, Computer Science Department, ANU, 1990.

[Neu95] Joachim Neubüser, *An invitation to computational group theory*, Groups '93 Galway/St. Andrews (C. M. Campbell, T. C. Hurley, E. F. Robertson, S. J. Tobin, and J. J. Ward, eds.), London Mathematical Society Lecture Note Series, vol. 212, Cambridge University Press, 1995, pp. 457–475.

[PW90] R[ichard] A. Parker and R[obert] A. Wilson, *The computer construction of matrix representations of finite groups over finite fields*, J. Symb. Comput. **9** (1990), no. 5–6, 583–590.

[S+97] Martin Schönert et al., *GAP 3.4, patchlevel 4*, Lehrstuhl D für Mathematik, Rheinisch-Westfälische Technische Hochschule, Aachen, 1997.

[Ser97] Ákos Seress, *An introduction to computational group theory*, Notices Amer. Math. Soc. **44** (1997), no. 6, 671–679.

[Soi93] L[eonard] H. Soicher, *GRAPE: a system for computing with graphs and groups*, Groups and Computation (Larry Finkelstein and William M. Kantor, eds.), DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, vol. 21, American Mathematical Society, Providence, RI, 1993, pp. 287–291.

E-mail address: ahulpke@dcs.st-and.ac.uk

E-mail address: sal@dcs.st-and.ac.uk

University of St. Andrews, School of Mathematical and Computational Sciences, The North Haugh, St. Andrews, Fife KY16 9SS, United Kingdom