MapClass

September 2018

Adam James
Kay Magaard
Sergey Shpectorov
Helmut Völklein

Sergey Shpectorov Email: s.shpectorov@bham.ac.uk

Copyright

Contents

Chapter 1

Introduction and Main Functions

This chapter provides an overview of the background material, and provides documentation for the main functions and data structures of the MapClass package.

1.1 Background Material

Let G be a finite group, let $C_1,\ldots,C_r$ be a collection of conjugacy classes in G. Let $\mathscr{E} = \mathscr{E}(G,g,(C_1,\ldots,C_r))$ denote the set of all tuples $\sigma = (\sigma_1,...,\sigma_{2g+r}) \in G^{2g+r}$ (for natural numbers g and r) of elements in G satisfying the relation

$$\prod_{i=1}^{g}[\sigma_i,\sigma_{g+i}]\prod_{i=1}^{r}\sigma_{2g+i} = 1$$

and such that $\sigma_{2g+k} \in C_k$. If the tuple also satisfies $\langle \sigma_1,\ldots,\sigma_{2g+r}\rangle = G$ it is said to be *generating*.

One may associate the elements of the tuple σ with the standard generators of the fundamental group of a compact connected surface S (genus g, r punctures). The mapping class group of S is naturally isomorphic to $Out(\pi_1(S))$ and so gives rise to an action on the fundamental group of S modulo inner automorphisms. This action can be transferred to an action on the set $\mathscr{E}$ (up to conjugation in G). The *mapping class orbits* are the orbits of $\mathscr{E}$ under this action.

The package can be used to compute the set $\mathscr{E}(G,g,(C_1,\ldots,C_r))$ and the corresponding partition into mapping class orbits for a given group G and a set of conjugacy classes $(C_1,...,C_r)$ (although the programs expect a tuple of class representatives). For an example application see Section 1.5. We call the tuple $(g;C_1,...,C_r)$ the signature. The package is an extension of the *Braid* package for GAP.

1.2 Overview of Main Functions

The following are the principal ways for calculating the mapping class orbits for a given signature and group. We require our groups to be permutation groups, and the tuple in question to have length at least two.

1.2.1 AllMCOrbits

▷ `AllMCOrbits(`*`group, genus, tuple`*`)` (function)

This function calculates the orbits for the given group, genus and tuple. This function is a wrapper for the function `AllMCOrbitsCore` (1.2.2), and so can make use of GAP's `OptionsStack`. The options are described in more detail in the documentation for `AllMCOrbitsCore` (1.2.2). We draw attention to two useful options: the `OutputStyle` and `SaveOrbit` options. The `SaveOrbit` option takes values of either `false` - in which case the orbit is not saved to a file - or it accepts a string that is the name of a directory in which the routine saves the orbits. See `AllMCOrbitsCore` (1.2.2) for more details on the saving process. The `OutputStyle` option controls the verbosity of the output of the function. It accepts three possible values:

- `"silent"` – the routine prints no output except in the case of an Error.
- `"single-line"` – the routine prints output to a single line. An intermediate output style for those who want some output but do not want to see all diagnostic output.
- `"full"` – the routine provides full diagnostic output.

1.2.2 AllMCOrbitsCore

▷ `AllMCOrbitsCore(`*`group, genus, tuple, partition, constant`*`)` (function)

This function calculates the orbits for the given group, genus and tuple, with the r branch points partitioned as in *`partition`*. It uses the given *`constant`* to determine how many of the tuples it want to account for before exiting. This function also make use of GAP's `OptionsStack` if one desires to alter how the algorithm runs. The following options and their defaults are given below.

Option Name	Default Value
`MaximumWeight`	40
`MinimumWeight`	20
`InitialWeight`	20
`BumpUp`	7
`KnockDown`	7
`InitialNumberOfRandomTuples`	20
`SaveOrbit`	`false`
`OutputStyle`	`"full"`

When trying to search for orbits it can be the case that the routine struggles to find a small orbit because of the low probability of randomly choosing a tuple in the orbit. To combat this problem the routine does not choose tuples entirely randomly but uses a weighted random selection to increase the probability of selecting tuples appearing in small tuples. To small subgroups of our group we have an associated weight. When a subgroup is generated by a tuple in our orbit frequently then we reduce its weight. Subgroups which do not appear often have their weight increased. The options `MaximumWeight`, `MinimumWeight`, `InitialWeight`,`BumpUp`, and `KnockDown`, control this subgroup weighting. Each option takes positive integer values. They play the following roles in the weighting process:

- `MaximumWeight` : The maximum weight that a subgroup can be.
- `MinimumWeight` : The minimum weight that a subgroup can be.
- `InitialWeight` : The weight that a new subgroup receives when added to to the list of small subgroups.

- `BumpUp` : The amount we increase the weight of a subgroup by when it does not appear frequently.
- `KnockDown` : The amount we decrease the weight of a subgroup by when it appears too frequently.

The default options were chosen experimentally and so it may be beneficial to tune these values for a specific case.

The option `InitialNumberOfRandomTuples` decides how many tuples the routine collects before trying to see if they are in some pre-existing orbit.

The option `SaveOrbit` which is by default `false` can be set to the name of a directory in which you want to save orbits. This option then saves the orbits to files in the folder with "`_name`". So for example if you wish to save your orbits into the file `_example` then you would run `AllMCOrbits(group, genus, tuple: SaveOrbit:="example");`. The orbits are then saved in orbits which are named numerically. Following on from the above example then the first orbit will be saved as "_example/0". The `OutputStyle` option controls the verbosity of the output. It accepts three possible values:

- `"silent"` - the routine prints no output except in the case of an Error.
- `"single-line"` - the routine output to a single line. An intermediate output style for those who want some output but do not want to see all diagnostic output.
- `"full"` - the routine provides full diagnostic output.

1.2.3 GeneratingMCOrbits

▷ `GeneratingMCOrbits(`*`group, genus, tuple`*`)` (function)

This calculates the orbits for the given arguments. Unlike the `AllMCOrbits` (1.2.1) function, `GeneratingMCOrbits` calculates only those orbits whose tuples generate the whole of our original group.

1.2.4 GeneratingMCOrbitsCore

▷ `GeneratingMCOrbitsCore(`*`group, genus, tuple, partition, constant`*`)` (function)

This calculates the orbits for the given arguments. Unlike the `AllMCOrbits` (1.2.1) function, `GeneratingMCOrbitsCore` calculates only those orbits whose tuples generate the whole of our original group. As with `AllMCOrbitsCore` (1.2.2), the argument *`partition`* must be a partition of the conjugacy classes represented in list form. We also have access to the full value of the options stack as in `AllMCOrbitsCore` (1.2.2).

1.2.5 MappingClassOrbit

▷ `MappingClassOrbit(`*`group, genus, principaltuple, partition, tuple`*`)` (function)

Returns: an orbit record for an orbit containing tuple or returns `fail`

Calculates the orbit of the *`tuple`* with respect to the given *`group`*, *`principaltuple`* and *`genus`*.

1.2.6 PrepareMinTree

▷ `PrepareMinTree(principaltuple, group, ourR, genus)` (function)

Returns: a record with two keys `MinimizationTree` and `MinimumSet`. If `record` is the returned record then `record.MinimizationTree` is the list encoding the tree used to help minimize tuples. The list `record.MinimumSet` is a list of minimal elements which is also used to speed up tuple minimization.

1.2.7 MinimizeTuple

▷ `MinimizeTuple(tuple, minimizationTree, minimumSet, numberOfGenerators)` (function)

Returns: the minimal tuple in the same orbit of *tuple*.

Take the minimisation data provided by `PrepareMinTree` (1.2.6) and minimizes the given *tuple*.

1.2.8 EasyMinimizeTuple

▷ `EasyMinimizeTuple(group, genus, tuple)` (function)

Returns: the minimal tuple in the same orbit as *tuple*.

1.2.9 IsInOrbit

▷ `IsInOrbit(orbit, tuple)` (function)

Returns: `True` if the *tuple* lies in the *orbit*.

1.2.10 FindTupleInOrbit

▷ `FindTupleInOrbit(orbit, tuple)` (function)

Returns: the index of *tuple* in `orbit.TupleTable` if in the *orbit*. If the tuple is not in *orbit* returns `fail`.

1.2.11 IsEqualOrbit

▷ `IsEqualOrbit(orbit1, orbit2)` (function)

Returns: `true` if the two orbits are equal else returns `fail`.

1.2.12 SelectTuple

▷ `SelectTuple(orbit, index)` (function)

Returns: the tuple `orbit.TupleTable[index].tuple`.

1.2.13 NumberGeneratingTuples

▷ `NumberGeneratingTuples(group, partition, tuple, genus)` (function)

Returns: the total number of possible generating tuples for the group and tuple.

1.2.14 TotalNumberTuples

▷ `TotalNumberTuples(`*`group, tuple, genus`*`)` (function)

Returns: the total number of tuples we have to account for.

1.2.15 CalculateTuplePartition

▷ `CalculateTuplePartition(`*`group, tuple`*`)` (function)

Returns: A partition of $1,\ldots,r$ where r is the length of the tuple.

The function returns a partition of $1,\ldots,r$ such that i and j lie in the same block if and only if the elements `tuple[i]` and `tuple[j]` are member of the same conjugacy class. The program currently requires that the elements of the tuple be ordered such that if `tuple[i]` and `tuple[j]` are in the same conjugacy class with $i \leq j$ then so is`tuple[k]` for all $i \leq k \leq j$.

1.2.16 RestoreOrbitFromFile

▷ `RestoreOrbitFromFile(`*`n, name[, path]`*`)` (function)

Returns: the n-th orbit record from the project with project `"name"`

By default the function searches the current working directory for the saved project folder and searches inside this for the n-th orbit. If no such orbit exists it returns `fail`. If an optional argument *`path`* is provided then it searches this path for a folder with the name specified (note that path expects a `Directory` object). If an orbit exists then it is returned as a record as outlined in the data structure section.

1.2.17 SaveOrbitToFile

▷ `SaveOrbitToFile(`*`orbit, n, name`*`)` (function)

Saves the orbit to filename "*`n`*" in the directory `'_name'`. The directory must already exist.

1.3 Key Data Structures

Many of the above functions require or return key data structures which we aim to document.

1.3.1 The Orbit Record

Many of the functions return or expect an orbit "object". This object is in fact record with the following values:

- `PrincipalFiniteGroup` - the finite group
- `OurG` - genus
- `OurR` - length of our primary tuple
- `OurN` - number of points on which our group acts
- `NumberOfGenerators` - 2 `OurG`+ `OurR`
- `OurFreeGroup` - a free group on `NumberOfGenerators` letters

- `AbsGens` - generators for `OurFreeGroup`
- `OurAlpha` - generators of `OurFreeGroup`corresponding to the α_i type loops in the fundamental group (the first g elements of `AbsGens`)
- `OurBeta` - elements of `OurFreeGroup` corresponding to β type loops
- `OurGamma` - generators of `OurFreeGroup` corresponding to the loops around branch points
- `TupleTable` - a table containing all the tuples in the orbit
- `HashLength`
- `Hash`
- `PrimeCode`
- `OurAction`
- `ActionOnOrbit`
- `MinimizationTree`- minimization structure
- `MinimumSet`- minimizaton structure

1.3.2 The Tuple Table

The tuple table is a list. Each element of the list is a record with the names, tuple and next. If `orbit` is an orbit object then `orbit.TupleTable[n].tuple` returns the tuple at index n of the tuple table.

1.4 A Sample Session

We demonstate how one might use the package.

Example

```
gap> group:=AlternatingGroup(5);
Alt( [ 1 .. 5 ] )
gap> tuple:=[ (1,2)(3,4), (1,2)(3,4), (1,2)(3,4) ];
[ (1,2)(3,4), (1,2)(3,4), (1,2)(3,4) ]
gap> orbits:=AllMCOrbits(group,1,tuple);;

Total Number of Tuples: 189120


Collecting 20 random tuples... done

Cleaning done; 20 random tuples remaining


Orbit 1:

Length=3072
Generating Tuple  =[ (1,2,4,5,3), (1,4,5,2,3), (1,2)(4,5),
(1,4)(2,3), (2,5)(3,4) ]
```

```
Generated subgroup size=60
Centralizer size=1
4800 tuples remaining
Cleaning current orbit...
Cleaning a list of 20 tuples
Random Tuples Remaining: 0
Cleaning done; 0 random tuples remaining


Collecting 20 random tuples... done

Cleaning orbit 1
Cleaning a list of 20 tuples
Random Tuples Remaining: 0

Cleaning done; 0 random tuples remaining


Collecting 40 random tuples... done

Cleaning orbit 1
Cleaning a list of 40 tuples
Random Tuples Remaining: 3

Cleaning done; 3 random tuples remaining


Orbit 2:

Length=32
Generating Tuple  =[ (1,4)(2,3), (1,2)(3,4), (1,4)(2,3), (1,2)(3,4),
(1,3)(2,4) ]
Generated subgroup size=4
Centralizer size=4
4320 tuples remaining
Cleaning current orbit...
Cleaning a list of 3 tuples
Random Tuples Remaining: 2
Cleaning done; 2 random tuples remaining


Orbit 3:

Length=72
Generating Tuple  =[ (1,5,2), (1,3,2), (1,2)(3,5), (1,3)(2,5),
(1,3)(2,5) ]
Generated subgroup size=12
Centralizer size=1
0 tuples remaining
Cleaning current orbit...
Cleaning a list of 2 tuples
Random Tuples Remaining: 0
```

```
Cleaning done; 0 random tuples remaining

gap> # Check we have as many orbits as it says...
gap> Length(orbits);
3
gap> # Inspect the first orbit..
gap> orb1:=orbits[1];;
gap> # How long is orb1?
gap> Length(orb1.TupleTable);
3072
gap> # Select element 42 ...
gap> SelectTuple(orb1, 42);
[ (1,3,4), (1,5,3,2,4), (1,5)(2,4), (1,2)(3,5), (2,3)(4,5) ]
gap> # Save the orbit to a file...
gap> SaveOrbitToFile(orb1, 1, "test");
gap> #If the folder doesn't exist we get an error..
gap> SaveOrbitToFile(orb1, 1, "foo");
AppendTo: cannot open '_foo/1' for output at
CallFuncList( APPEND_TO, arg );
gap> #
gap> # Now we try find generating orbits
gap> group:=SymmetricGroup(5);
Sym( [ 1 .. 5 ] )
gap> # And we will save them using the `SaveOrbit` option
gap> GeneratingMCOrbits(group,1,tuple : SaveOrbit:="example");;

Total Number of Tuples: 607680



Collecting 20 generating tuples .. done

Cleaning done; 20 random tuples remaining


Orbit 1:
Length=5064
Generating Tuple  =[ (1,3,2,5), (2,4,3), (1,4)(3,5), (1,3)(2,5),
(1,4)(3,5) ]
Generated subgroup size=120
Saving orbit to file _example/0
Centralizer size=1
0 tuples remaining
Cleaning current orbit...
Cleaning a list of 20 tuples
Random Tuples Remaining: 0
Cleaning done; 0 random tuples remaining

gap> generatingorbits:=last;;
gap> # How many generating orbits are there?
gap> Length(generatingorbits);
1
```

```
gap> # Is this orbit equal to the other one we found earlier
gap> IsEqualOrbit(orb1, generatingorbits[1]);
fail
gap> # We can reload the orbits...
gap> orb2:=RestoreOrbitFromFile(0, "example");;
gap> Length(orb2.TupleTable);
5064
```

1.5 An Application

This section describes an application of the package.

Let X be a compact Riemann surface of genus g and $f : X \to \mathbb{P}^1\mathbb{C}$ be a meromorphic function of degree n. Let B be the set of branch points for f and fix a basepoint $b_0 \in \mathbb{P}^1\mathbb{C} - B$. The fundamental group $\pi_1(\mathbb{P}^1\mathbb{C} - B, b_0)$ acts transitively on the fibre $f^{-1}(b_0)$ and this corresponds to a representation

$$f^* : \pi_1(\mathbb{P}^1\mathbb{C} - B, b_0) \to S_n$$

The image of f^* is called the *monodromy group* of (X, f). The fundamental group of the punctured Riemann sphere is generated by the loops that wind around the points in B. Label the branch points $b_1, ..., b_r$ and let τ_i be the image under f^* of the loop, $\gamma_i \in \pi_1(\mathbb{P}^1\mathbb{C} - B)$, that winds once around the point $b_i \in B$. Therefore,

$$\langle \tau_1, ..., \tau_r \rangle = G$$

and

$$\tau_1 \cdots \tau_r = 1$$

Moreover by the Riemann-Hurwitz formula

$$2(n + g - 1) = \sum_1^r ind(\tau_i)$$

where the $ind(\tau_i)$ is the minimal number of factors to express τ_i as a product of transpositions. A set $t_1, \ldots, t_r$ of elements of S_n satisfying the Riemann-Hurwitz formula, the product-one condition, and generating some transitive subgroup G of S_n is called a *genus g generating system* for G. Therefore to the meromorphic function (X, f) there is an associated genus g system. In fact the conjugacy classes of the elements τ_i are also determined by f – the collection of conjugacy classes is sometimes called the *ramification type* of f. On the other hand for every genus g generating system, $t = (\tau_1, \ldots, \tau_n)$ for G there is Riemann surface of genus g and a meromorphic function with associated generating system t – this result is known as *Riemann's Existence Theorem*.

The question we hope to use our package to answer is: For a given finite group G how many meromorphic maps with monodromy group G are there? It can be shown – see [Völ96] for example – that determining whether two genus g coverings are equivalent corresponds to determining whether their associated genus g systems are in the same mapping class orbit (most literature would refer to mapping class orbits as braid orbits in this case - this is because of the equivalence between the mapping class group of a punctured disc and the braid groups [Bir75]).

Thus for a finite group G we can answer the above principal question using the following process:

- For a given finite group G the work of Breuer [Bre00] can be used to calculate all possible ramification types.

- Pick a tuple, $C = (c_1, ..., c_r)$, of representative elements of the conjugacy classes which correspond to a chosen ramification type as calculated in the previous step.
- Use the function `GeneratingMCOrbits(G, 0, [c1,...,cr])` to calculate the number of mapping class orbits. Note that the genus argument is 0 because this is the genus of $\mathbb{P}^1\mathbb{C}$.

For more information on this process and the underlying theory see [MSSV02] and [Völ96].

Appendix A

Installation

To Install the package place the "MapClass" folder into your GAP system's 'pkg' directory. If you do not have permission to modify the gap package then the package can be included by appending a local directory to GAP's root directory using the '-l' flag. For more information on GAP's root directory process try '?GAP root' in a GAP session. Or see the online help at http://www.gap-system.org/Manuals/doc/htm/ref/CHAP009.htm#SECT002 To Load the package simply type 'LoadPackage("mapclass");'. If the load has been successful the package banner will be shown.

References

[Bir75] J. S. Birman. *Braids, links, and mapping class groups*. Princeton University Press, Princeton, N. J., 1975. Based on lecture notes by James Cannon. 12

[Bre00] T. Breuer. *Characters and automorphism groups of compact Riemann surfaces*, volume 280. Cambridge University Press, 2000. 12

[MSSV02] K. Magaard, T. Shaska, S. Shpectorov, and H. Völklein. The locus of curves with prescribed automorphism group. *Communications in arithmetic fundamental groups (Kyoto, 1999/2000)*, 2002. 13

[Völ96] H. Völklein. *Groups as Galois groups*, volume 53 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1996. An introduction. 12, 13

Index