Computing braid orbits

A. James and S. Shpectorov

School of Mathematics, University of Birmingham

ICMS Sigma Workshop, 15th October 2010

Origin

Origin

Consider a meromorphic function $f : \mathbb{P}^1 \to \mathbb{P}^1$. Associated with f, there is its monodromy group $G = G(f)$, which is a permutation group on n points, where n is the degree of f.

Origin

Consider a meromorphic function $f : \mathbb{P}^1 \to \mathbb{P}^1$. Associated with f, there is its monodromy group $G = G(f)$, which is a permutation group on n points, where n is the degree of f. Hence G is always finite, but its size is unlimited

Origin

Consider a meromorphic function $f : \mathbb{P}^1 \to \mathbb{P}^1$. Associated with f, there is its monodromy group $G = G(f)$, which is a permutation group on n points, where n is the degree of f. Hence G is always finite, but its size is unlimited

Guralnick–Thompson Conjecture

Every composition factor of G is either the alternating group A_k for some $k \geq 5$, or it belongs to a finite list of exceptions.

Origin

Consider a meromorphic function $f : \mathbb{P}^1 \to \mathbb{P}^1$. Associated with f, there is its monodromy group $G = G(f)$, which is a permutation group on n points, where n is the degree of f. Hence G is always finite, but its size is unlimited

Guralnick–Thompson Conjecture

Every composition factor of G is either the alternating group A_k for some $k \geq 5$, or it belongs to a finite list of exceptions.

This was later generalized to the case of a meromorphic function on a Rieman surface of bounded genus g

Origin

Consider a meromorphic function $f : \mathbb{P}^1 \to \mathbb{P}^1$. Associated with f, there is its monodromy group $G = G(f)$, which is a permutation group on n points, where n is the degree of f. Hence G is always finite, but its size is unlimited

Guralnick–Thompson Conjecture

Every composition factor of G is either the alternating group A_k for some $k \geq 5$, or it belongs to a finite list of exceptions.

This was later generalized to the case of a meromorphic function on a Rieman surface of bounded genus g (same conclusion, but the list of exceptions grows with g).

Origin

Consider a meromorphic function $f : \mathbb{P}^1 \to \mathbb{P}^1$. Associated with f, there is its monodromy group $G = G(f)$, which is a permutation group on n points, where n is the degree of f. Hence G is always finite, but its size is unlimited

Guralnick–Thompson Conjecture

Every composition factor of G is either the alternating group A_k for some $k \geq 5$, or it belongs to a finite list of exceptions.

This was later generalized to the case of a meromorphic function on a Rieman surface of bounded genus g (same conclusion, but the list of exceptions grows with g).
This conjecture was finally proved (for arbitrary g) by Frohardt and Magaard in 2001.

Origin

Consider a meromorphic function $f : \mathbb{P}^1 \to \mathbb{P}^1$. Associated with f, there is its monodromy group $G = G(f)$, which is a permutation group on n points, where n is the degree of f. Hence G is always finite, but its size is unlimited

Guralnick–Thompson Conjecture

Every composition factor of G is either the alternating group A_k for some $k \geq 5$, or it belongs to a finite list of exceptions.

This was later generalized to the case of a meromorphic function on a Rieman surface of bounded genus g (same conclusion, but the list of exceptions grows with g).
This conjecture was finally proved (for arbitrary g) by Frohardt and Magaard in 2001.
Magaard also wanted to determine the complete list of exceptional simple groups at least for $g = 0$, but if possible also for other small g.

Braid orbits

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.
Generating tuples are tuples $(g_1, \ldots, g_r)$, where $G = \langle g_1, \ldots, g_r \rangle$ and also $g_1 \cdot \ldots \cdot g_r = 1$.

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.
Generating tuples are tuples $(g_1, \ldots, g_r)$, where $G = \langle g_1, \ldots, g_r \rangle$ and also $g_1 \cdot \ldots \cdot g_r = 1$.
The action of the *braid group on r strands*, B_r, on the set of generating tuples of G is given by:

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.
Generating tuples are tuples $(g_1, \ldots, g_r)$, where $G = \langle g_1, \ldots, g_r \rangle$ and also $g_1 \cdot \ldots \cdot g_r = 1$.
The action of the *braid group on* r *strands*, B_r, on the set of generating tuples of G is given by:

$$\tau_i : (g_1, \ldots, g_i, g_{i+1}, \ldots, g_r) \to (g_1, \ldots, g_{i+1}, g_{i+1}^{-1} g_i g_{i+1}, \ldots, g_r),$$

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.
Generating tuples are tuples $(g_1, \ldots, g_r)$, where $G = \langle g_1, \ldots, g_r \rangle$ and also $g_1 \cdot \ldots \cdot g_r = 1$.
The action of the *braid group on r strands*, B_r, on the set of generating tuples of G is given by:

$$\tau_i : (g_1, \ldots, g_i, g_{i+1}, \ldots, g_r) \to (g_1, \ldots, g_{i+1}, g_{i+1}^{-1} g_i g_{i+1}, \ldots, g_r),$$

where $\tau_1, \ldots, \tau_{r-1}$ are elementary braids.

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.
Generating tuples are tuples $(g_1, \ldots, g_r)$, where $G = \langle g_1, \ldots, g_r \rangle$ and also $g_1 \cdot \ldots \cdot g_r = 1$.
The action of the *braid group on r strands*, B_r, on the set of generating tuples of G is given by:

$$\tau_i : (g_1, \ldots, g_i, g_{i+1}, \ldots, g_r) \to (g_1, \ldots, g_{i+1}, g_{i+1}^{-1} g_i g_{i+1}, \ldots, g_r),$$

where $\tau_1, \ldots, \tau_{r-1}$ are elementary braids. The *elementary braid* τ_i takes the ith strand *over* the next strand.

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.
Generating tuples are tuples $(g_1, \ldots, g_r)$, where $G = \langle g_1, \ldots, g_r \rangle$ and also $g_1 \cdot \ldots \cdot g_r = 1$.
The action of the *braid group on r strands*, B_r, on the set of generating tuples of G is given by:

$$\tau_i : (g_1, \ldots, g_i, g_{i+1}, \ldots, g_r) \to (g_1, \ldots, g_{i+1}, g_{i+1}^{-1} g_i g_{i+1}, \ldots, g_r),$$

where $\tau_1, \ldots, \tau_{r-1}$ are elementary braids. The *elementary braid* τ_i takes the ith strand *over* the next strand. The τ_i generate B_r.

Braid orbits

Different (from the group theory point of view) cases of functions f correspond to the *braid orbits* on the generating tuples of the permutation group G.
Generating tuples are tuples $(g_1, \ldots, g_r)$, where $G = \langle g_1, \ldots, g_r \rangle$ and also $g_1 \cdot \ldots \cdot g_r = 1$.
The action of the *braid group on r strands*, B_r, on the set of generating tuples of G is given by:

$$\tau_i : (g_1, \ldots, g_i, g_{i+1}, \ldots, g_r) \to (g_1, \ldots, g_{i+1}, g_{i+1}^{-1} g_i g_{i+1}, \ldots, g_r),$$

where $\tau_1, \ldots, \tau_{r-1}$ are elementary braids. The *elementary braid* τ_i takes the ith strand *over* the next strand. The τ_i generate B_r.
Note that the tuple on the right is again a generating tuple of G, so we indeed have an action of B_r and orbits.

BRAID

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.
After the lecture SSh told him that this is much better done by computer, and within a week they got together and the first unsophisticated version of BRAID was written.

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.
After the lecture SSh told him that this is much better done by computer, and within a week they got together and the first unsophisticated version of BRAID was written.
Within a short time, BRAID went through several versions, each improving performance by a large factor.

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.
After the lecture SSh told him that this is much better done by computer, and within a week they got together and the first unsophisticated version of BRAID was written.
Within a short time, BRAID went through several versions, each improving performance by a large factor.
It has two main routines:

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.
After the lecture SSh told him that this is much better done by computer, and within a week they got together and the first unsophisticated version of BRAID was written.
Within a short time, BRAID went through several versions, each improving performance by a large factor.
It has two main routines:

- `BraidOrbit` computes a braid orbit starting from a single tuple.

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.
After the lecture SSh told him that this is much better done by computer, and within a week they got together and the first unsophisticated version of BRAID was written.
Within a short time, BRAID went through several versions, each improving performance by a large factor.
It has two main routines:

- `BraidOrbit` computes a braid orbit starting from a single tuple.
- `BraidOrbits` computes *all* braid orbits where the entries g_i are selected within the given set of r conjugacy classes $C_1, \dots, C_r$ of elements of G.

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.
After the lecture SSh told him that this is much better done by computer, and within a week they got together and the first unsophisticated version of BRAID was written.
Within a short time, BRAID went through several versions, each improving performance by a large factor.
It has two main routines:

- `BraidOrbit` computes a braid orbit starting from a single tuple.
- `BraidOrbits` computes *all* braid orbits where the entries g_i are selected within the given set of r conjugacy classes $C_1, \ldots, C_r$ of elements of G.

`BraidOrbits` generates random tuples with product 1 condition, but possibly nongenerating, and calls `BraidOrbit` to construct new orbits until the total reaches the *structure constant*.

BRAID

Around 1999-2000 SSh attended a talk by Kay Magaard on the Guralnick-Thompson conjecture, where Kay mentioned *hand computation* of braid orbits.
After the lecture SSh told him that this is much better done by computer, and within a week they got together and the first unsophisticated version of BRAID was written.
Within a short time, BRAID went through several versions, each improving performance by a large factor.
It has two main routines:

- `BraidOrbit` computes a braid orbit starting from a single tuple.
- `BraidOrbits` computes *all* braid orbits where the entries g_i are selected within the given set of r conjugacy classes $C_1, \ldots, C_r$ of elements of G.

`BraidOrbits` generates random tuples with product 1 condition, but possibly nongenerating, and calls `BraidOrbit` to construct new orbits until the total reaches the *structure constant*. The latter is precomputed from the character table of G.

Hurwitz loci

Helmut Völklein joined in as a customer and then also co-author and he explained what the braid orbits were classifying—Hurwitz loci.

Hurwitz loci

Helmut Völklein joined in as a customer and then also co-author and he explained what the braid orbits were classifying—Hurwitz loci. *Hurwitz loci* are the connected components of the Hurwitz space $\mathcal{H}_g$, the moduli space of all pairs (X, G), where X is compact Riemann surface of genus g and G is a finite group of isotopic transformations of X.

Hurwitz loci

Helmut Völklein joined in as a customer and then also co-author and he explained what the braid orbits were classifying—Hurwitz loci. *Hurwitz loci* are the connected components of the Hurwitz space $\mathcal{H}_g$, the moduli space of all pairs (X, G), where X is compact Riemann surface of genus g and G is a finite group of isotopic transformations of X. Clearly, on every locus, G remains the same as an abstract finite group, so the loci can be classified according which group they involve.

Hurwitz loci

Helmut Völklein joined in as a customer and then also co-author and he explained what the braid orbits were classifying—Hurwitz loci. *Hurwitz loci* are the connected components of the Hurwitz space $\mathcal{H}_g$, the moduli space of all pairs (X, G), where X is compact Riemann surface of genus g and G is a finite group of isotopic transformations of X. Clearly, on every locus, G remains the same as an abstract finite group, so the loci can be classified according which group they involve. We will also use the same term "Hurwitz loci" for the images of Hurwitz loci in the moduli space $\mathcal{M}_g$ of all compact Riemann surfaces of genus g.

Hurwitz loci

Helmut Völklein joined in as a customer and then also co-author and he explained what the braid orbits were classifying—Hurwitz loci. *Hurwitz loci* are the connected components of the Hurwitz space $\mathcal{H}_g$, the moduli space of all pairs (X, G), where X is compact Riemann surface of genus g and G is a finite group of isotopic transformations of X. Clearly, on every locus, G remains the same as an abstract finite group, so the loci can be classified according which group they involve. We will also use the same term "Hurwitz loci" for the images of Hurwitz loci in the moduli space $\mathcal{M}_g$ of all compact Riemann surfaces of genus g. This helps picturing the Hurwitz loci as forming a structure under inclusion where the loci for larger groups G are contained in the loci for their subgroups.

Hurwitz loci

Helmut Völklein joined in as a customer and then also co-author and he explained what the braid orbits were classifying—Hurwitz loci. *Hurwitz loci* are the connected components of the Hurwitz space $\mathcal{H}_g$, the moduli space of all pairs (X, G), where X is compact Riemann surface of genus g and G is a finite group of isotopic transformations of X. Clearly, on every locus, G remains the same as an abstract finite group, so the loci can be classified according which group they involve. We will also use the same term "Hurwitz loci" for the images of Hurwitz loci in the moduli space $\mathcal{M}_g$ of all compact Riemann surfaces of genus g. This helps picturing the Hurwitz loci as forming a structure under inclusion where the loci for larger groups G are contained in the loci for their subgroups.
The braid orbits, as defined above, classify the loci for the given group G, but only those where the *orbit genus* g_0, that is, the genus of the orbit curve Y/G, is zero (and so $Y = \mathbb{P}^1$).

Application: Hurwitz loci for large groups G

Application: Hurwitz loci for large groups G

We say that the isotopies group G is *large* if $|G| > 4(g-1)$.

Application: Hurwitz loci for large groups G

We say that the isotopies group G is *large* if $|G| > 4(g-1)$. This condition guarantees that the orbit genus is zero, hence BRAID could be used.

Application: Hurwitz loci for large groups G

We say that the isotopies group G is *large* if $|G| > 4(g-1)$. This condition guarantees that the orbit genus is zero, hence BRAID could be used.
We determined (2002) all Hurwitz loci for large groups G with $g \leq 10$. Tony Shaska joined us in this project and he additionally wrote the exact equations of curves in each locus for $g = 3$.

Application: Hurwitz loci for large groups G

We say that the isotopies group G is *large* if $|G| > 4(g-1)$. This condition guarantees that the orbit genus is zero, hence BRAID could be used.
We determined (2002) all Hurwitz loci for large groups G with $g \leq 10$. Tony Shaska joined us in this project and he additionally wrote the exact equations of curves in each locus for $g = 3$.
This computation was based on the previous work of Breuer, who determined all groups G that can act on compact Riemann surfaces with $2 \leq g < 50$ and also determined all corresponding *types* $(C_1, \ldots, C_r)$.

Arbitrary orbit genus?

Arbitrary orbit genus?

Of course, it would be great to be able to do the general case of arbitrary orbit genus.

Arbitrary orbit genus?

Of course, it would be great to be able to do the general case of arbitrary orbit genus. However, not all necessary ingredients were readily available and so they needed to be worked out.

Arbitrary orbit genus?

Of course, it would be great to be able to do the general case of arbitrary orbit genus. However, not all necessary ingredients were readily available and so they needed to be worked out.
The changes involved:

Arbitrary orbit genus?

Of course, it would be great to be able to do the general case of arbitrary orbit genus. However, not all necessary ingredients were readily available and so they needed to be worked out.

The changes involved:

- Standard tuples.

Arbitrary orbit genus?

Of course, it would be great to be able to do the general case of arbitrary orbit genus. However, not all necessary ingredients were readily available and so they needed to be worked out.
The changes involved:

- Standard tuples.
- Mapping class groups.

Arbitrary orbit genus?

Of course, it would be great to be able to do the general case of arbitrary orbit genus. However, not all necessary ingredients were readily available and so they needed to be worked out.
The changes involved:

- Standard tuples.
- Mapping class groups.
- Group–subgroup correspondence.

Standard tuples

Standard tuples

First of all, the generating tuples $(g_1, \ldots, g_r)$ had to be substituted with more complicatedly looking *standard tuples*.

Standard tuples

First of all, the generating tuples $(g_1, \ldots, g_r)$ had to be substituted with more complicatedly looking *standard tuples*. These are tuples

$$(a_1, \ldots, a_{g_0}, b_1, \ldots, b_{g_0}, c_1, \ldots, c_r),$$

which again (1) must generate G, and (2) instead of the product 1 condition, must satisfy:

$$[a_1, b_1] \cdots [a_{g_0}, b_{g_0}] c_1 \cdots c_r = 1.$$

Standard tuples

First of all, the generating tuples $(g_1, \ldots, g_r)$ had to be substituted with more complicatedly looking *standard tuples*. These are tuples

$$(a_1, \ldots, a_{g_0}, b_1, \ldots, b_{g_0}, c_1, \ldots, c_r),$$

which again (1) must generate G, and (2) instead of the product 1 condition, must satisfy:

$$[a_1, b_1] \cdots [a_{g_0}, b_{g_0}] c_1 \cdots c_r = 1.$$

Here $[a, b] = a^{-1} b^{-1} ab$ is the *commutator* of a and b.

Standard tuples

First of all, the generating tuples $(g_1, \ldots, g_r)$ had to be substituted with more complicatedly looking *standard tuples*. These are tuples

$$(a_1, \ldots, a_{g_0}, b_1, \ldots, b_{g_0}, c_1, \ldots, c_r),$$

which again (1) must generate G, and (2) instead of the product 1 condition, must satisfy:

$$[a_1, b_1] \cdots [a_{g_0}, b_{g_0}] c_1 \cdots c_r = 1.$$

Here $[a, b] = a^{-1} b^{-1} ab$ is the *commutator* of a and b.
This comes from the consideration of the fundamental group of $\hat{Y}$, which is the orbit curve Y with all ramified points removed (punctured).

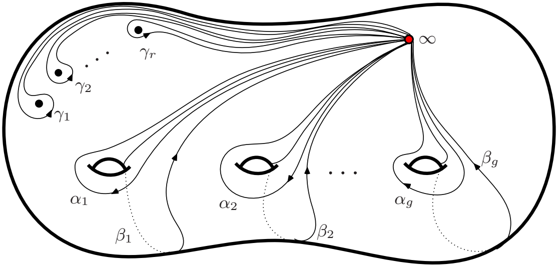
Fundamenta group of $\hat{Y}$

Fundamenta group of $\hat{Y}$

The following elements of $\pi_1(\hat{Y})$ are called its standard generators.

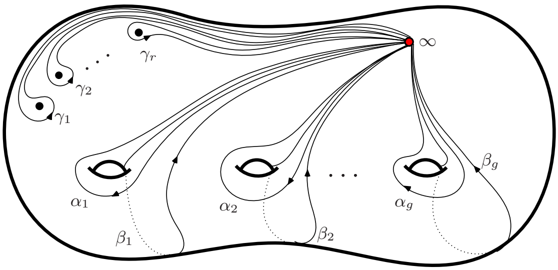
Fundamenta group of $\hat{Y}$

The following elements of $\pi_1(\hat{Y})$ are called its standard generators.



Fundamenta group of $\hat{Y}$

The following elements of $\pi_1(\hat{Y})$ are called its standard generators.

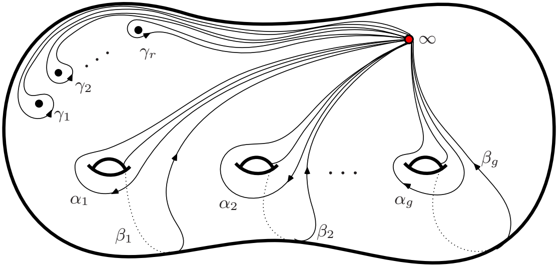


These loops satisfy

$$[\alpha_1, \beta_1] \cdots [\alpha_{g_0}, \beta_{g_0}] \gamma_1 \cdots \gamma_r = 1.$$

Fundamenta group of $\hat{Y}$

The following elements of $\pi_1(\hat{Y})$ are called its standard generators.



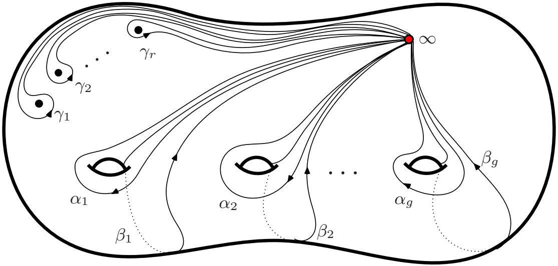
These loops satisfy

$$[\alpha_1, \beta_1] \cdots [\alpha_{g_0}, \beta_{g_0}] \gamma_1 \cdots \gamma_r = 1.$$

The standard tuples in G are simply the images in G of these standard generators.

Fundamenta group of $\hat{Y}$

The following elements of $\pi_1(\hat{Y})$ are called its standard generators.



These loops satisfy

$$[\alpha_1, \beta_1] \cdots [\alpha_{g_0}, \beta_{g_0}]\gamma_1 \cdots \gamma_r = 1.$$

The standard tuples in G are simply the images in G of these standard generators. They form an orbit under the action of the *mapping class group* $Mod_{g,r+1}$ of $\hat{Y} - \{\infty\}$.

Mapping class group

Mapping class group

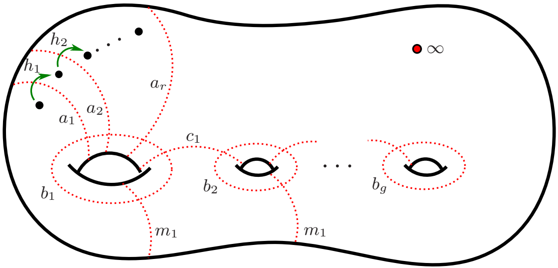
The *mapping class group* is the group of self homeomorphisms taken up to isotopy.

Mapping class group

The *mapping class group* is the group of self homeomorphisms taken up to isotopy.
The following set of generators was taken from a paper by Labruere and Paris.

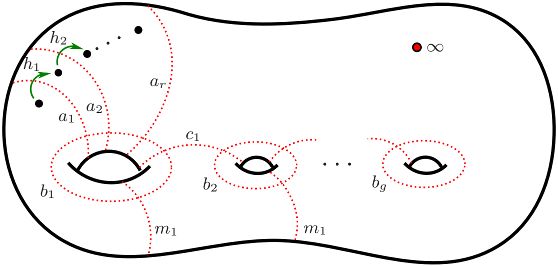
Mapping class group

The *mapping class group* is the group of self homeomorphisms taken up to isotopy.
The following set of generators was taken from a paper by Labruere and Paris.



Mapping class group

The *mapping class group* is the group of self homeomorphisms taken up to isotopy.
The following set of generators was taken from a paper by Labruere and Paris.



The arrows indicate half-twists (braid twists) and red dotted lines indicate Dehn twists around the suitable loops.

Action

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$.

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$. The following only shows the nontrivial part of the action.

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$. The following only shows the nontrivial part of the action.

- The action of a_i:

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$. The following only shows the nontrivial part of the action.

- The action of a_i:
  - $T_{a_i}(\beta_i) = \beta_i \alpha_i^{-1}$.

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$. The following only shows the nontrivial part of the action.

- The action of a_i:
  - $T_{a_i}(\beta_i) = \beta_i \alpha_i^{-1}$.
- The action of c_{i-1}:

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$. The following only shows the nontrivial part of the action.

- The action of a_i:
  - $T_{a_i}(\beta_i) = \beta_i \alpha_i^{-1}$.
- The action of c_{i-1}:
  - $T_{c_{i-1}}(\alpha_i) = \alpha_i \beta_{i-1} \alpha_i^{-1} \beta_i^{-1} \alpha_i$.

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$. The following only shows the nontrivial part of the action.

- The action of a_i:
  - $T_{a_i}(\beta_i) = \beta_i \alpha_i^{-1}$.
- The action of c_{i-1}:
  - $T_{c_{i-1}}(\alpha_i) = \alpha_i \beta_{i-1} \alpha_i^{-1} \beta_i^{-1} \alpha_i$.
- The action of c_i:

Action

We computed (SSh, also recomputed by AJ) the action of these generators of $Mod_{g,r+1}$ on the standard generators of $\pi_(\hat{Y}, \infty)$. The following only shows the nontrivial part of the action.

- The action of a_i:
  - $T_{a_i}(\beta_i) = \beta_i \alpha_i^{-1}$.
- The action of c_{i-1}:
  - $T_{c_{i-1}}(\alpha_i) = \alpha_i \beta_{i-1} \alpha_i^{-1} \beta_i^{-1} \alpha_i$.
- The action of c_i:
  - $T_{c_i}(\alpha_i) = \alpha_{i+1}^{-1} \beta_{i+1} \alpha_{i+1} \beta_i^{-1} \alpha_i$.
  - $T_{c_i}(\beta_i) = \alpha_{i+1}^{-1} \beta_{i+1} \alpha_{i+1} \beta_i^{-1} \alpha_{i+1}^{-1} \beta_{i+1}^{-1} \alpha_{i+1}$.

Action

Action

- The action of f_i:

Action

- The action of f_i:
  - $T_{f_i}(\alpha_1) = \alpha_1 \gamma_n^{-1} \cdots \gamma_{i+1}^{-1} \alpha_1^{-1} \beta_1^{-1} \alpha_1$.
  - $T_{f_i}(\gamma_j) = (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1) \gamma_j (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1)^{-1}$.

Action

- The action of f_i:
  - $T_{f_i}(\alpha_1) = \alpha_1 \gamma_n^{-1} \cdots \gamma_{i+1}^{-1} \alpha_1^{-1} \beta_1^{-1} \alpha_1$.
  - $T_{f_i}(\gamma_j) = (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1) \gamma_j (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1)^{-1}$.
- The action of m_i:

Action

- The action of f_i:
  - $T_{f_i}(\alpha_1) = \alpha_1 \gamma_n^{-1} \cdots \gamma_{i+1}^{-1} \alpha_1^{-1} \beta_1^{-1} \alpha_1$.
  - $T_{f_i}(\gamma_j) = (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1) \gamma_j (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1)^{-1}$.
- The action of m_i:
  - $T_{m_i}(\alpha_i) = \beta_i^{-1} \alpha_i$

Action

- The action of f_i:
  - $T_{f_i}(\alpha_1) = \alpha_1 \gamma_n^{-1} \cdots \gamma_{i+1}^{-1} \alpha_1^{-1} \beta_1^{-1} \alpha_1$.
  - $T_{f_i}(\gamma_j) = (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1) \gamma_j (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1)^{-1}$.
- The action of m_i:
  - $T_{m_i}(\alpha_i) = \beta_i^{-1} \alpha_i$

This action on the standard generators of $\pi_1(\hat{Y}, \infty)$ directly translates to an action of $Mod_{g,r+1}$ on the standard tuples in G,

Action

- The action of f_i:
  - $T_{f_i}(\alpha_1) = \alpha_1 \gamma_n^{-1} \cdots \gamma_{i+1}^{-1} \alpha_1^{-1} \beta_1^{-1} \alpha_1$.
  - $T_{f_i}(\gamma_j) = (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1) \gamma_j (\gamma_{i+1} \cdots \gamma_n \alpha_1^{-1} \beta_1^{-1} \alpha_1)^{-1}$.
- The action of m_i:
  - $T_{m_i}(\alpha_i) = \beta_i^{-1} \alpha_i$

This action on the standard generators of $\pi_1(\hat{Y}, \infty)$ directly translates to an action of $Mod_{g,r+1}$ on the standard tuples in G, giving us the *mapping class orbits* that classify the arbitrary Hurwitz loci, just like the braid orbits classify the Hurwitz loci with orbit genus zero.

Reduction to subgroup

Magaard and SSh also developed an algorithm that, given a standard tuple in G, computes a standard tuple in any subgroup $H \leq G$ corresponding to the action of H on the same Riemann surface X.

Reduction to subgroup

Magaard and SSh also developed an algorithm that, given a standard tuple in G, computes a standard tuple in any subgroup $H \leq G$ corresponding to the action of H on the same Riemann surface X. This allows us to determine the inclusions between Hurwitz loci of different groups.

Application: Hurwitz loci for $g \leq 16$

Application: Hurwitz loci for $g \leq 16$

Using these tools we determined the complete structure (with inclusions) of *all* Hurwitz loci for $g \leq 16$.

Application: Hurwitz loci for $g \leq 16$

Using these tools we determined the complete structure (with inclusions) of *all* Hurwitz loci for $g \leq 16$. The tables are too big to be shown here but they are available upon request.

Application: Hurwitz loci for $g \leq 16$

Using these tools we determined the complete structure (with inclusions) of *all* Hurwitz loci for $g \leq 16$. The tables are too big to be shown here but they are available upon request.
They haven't been independently verified.

Application: Hurwitz loci for $g \leq 16$

Using these tools we determined the complete structure (with inclusions) of *all* Hurwitz loci for $g \leq 16$. The tables are too big to be shown here but they are available upon request.
They haven't been independently verified.
However, for $g = 3$ and 4, they were compared with the published "hand-made" results.

Application: Hurwitz loci for $g \leq 16$

Using these tools we determined the complete structure (with inclusions) of *all* Hurwitz loci for $g \leq 16$. The tables are too big to be shown here but they are available upon request.
They haven't been independently verified.
However, for $g = 3$ and 4, they were compared with the published "hand-made" results. In fact, we corrected a few errors there.

Application: Hurwitz loci for $g \leq 16$

Using these tools we determined the complete structure (with inclusions) of *all* Hurwitz loci for $g \leq 16$. The tables are too big to be shown here but they are available upon request.
They haven't been independently verified.
However, for $g = 3$ and 4, they were compared with the published "hand-made" results. In fact, we corrected a few errors there.
Another interesting feature is that we found many instances of equal loci for different groups.

Surface braid group

Surface braid group

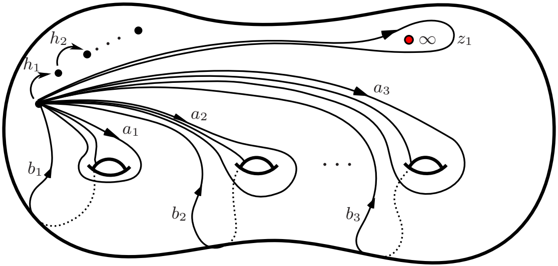
AJ also computed the action of the *surface braid group*, which is a normal subgroup of $Mod_{g,r+1}$.

Surface braid group

AJ also computed the action of the *surface braid group*, which is a normal subgroup of $Mod_{g,r+1}$.
The generators of the surface braid group are as follows.

Surface braid group

AJ also computed the action of the *surface braid group*, which is a normal subgroup of $Mod_{g,r+1}$.
The generators of the surface braid group are as follows.



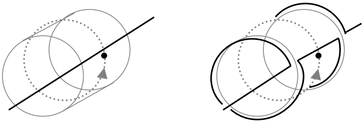
Surface braid group

Surface braid group

Each generator is a product of two Dehn twists.

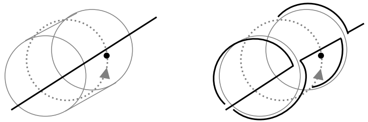
Surface braid group

Each generator is a product of two Dehn twists.



Surface braid group

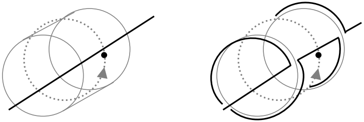
Each generator is a product of two Dehn twists.



First, around one boundary loop of a thin annular neighbourhood of the main loop,

Surface braid group

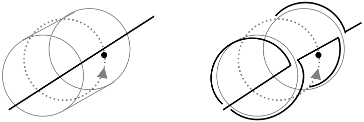
Each generator is a product of two Dehn twists.



First, around one boundary loop of a thin annular neighbourhood of the main loop, then around the second boundary loop in the opposite direction (inverse).

Surface braid group

Each generator is a product of two Dehn twists.



First, around one boundary loop of a thin annular neighbourhood of the main loop, then around the second boundary loop in the opposite direction (inverse). This product is trivial when the punctures are filled.

Larger orbits

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).
We are working on three ideas that will hopefully allow us to compute much larger orbits.

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).

We are working on three ideas that will hopefully allow us to compute much larger orbits.

- Orbit computation is well suited for use of parallel processors.

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).

We are working on three ideas that will hopefully allow us to compute much larger orbits.

- Orbit computation is well suited for use of parallel processors. We did a sample computation of an orbit of size 1.2 million using SCSCP package.

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).

We are working on three ideas that will hopefully allow us to compute much larger orbits.

- Orbit computation is well suited for use of parallel processors. We did a sample computation of an orbit of size 1.2 million using SCSCP package.
- We can use the smaller surface braid group action to only enumerate a part of the mapping class orbit.

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).
We are working on three ideas that will hopefully allow us to compute much larger orbits.

- Orbit computation is well suited for use of parallel processors. We did a sample computation of an orbit of size 1.2 million using SCSCP package.
- We can use the smaller surface braid group action to only enumerate a part of the mapping class orbit. Gives a factor of hundreds or even thousands.

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).
We are working on three ideas that will hopefully allow us to compute much larger orbits.

- Orbit computation is well suited for use of parallel processors. We did a sample computation of an orbit of size 1.2 million using SCSCP package.
- We can use the smaller surface braid group action to only enumerate a part of the mapping class orbit. Gives a factor of hundreds or even thousands.
- Jason Wang is trying to program an approach based on splitting the tuple.

Larger orbits

Currently, we can compute orbits up to about a million (tuples up to conjugation in G).
We are working on three ideas that will hopefully allow us to compute much larger orbits.

- Orbit computation is well suited for use of parallel processors. We did a sample computation of an orbit of size 1.2 million using SCSCP package.
- We can use the smaller surface braid group action to only enumerate a part of the mapping class orbit. Gives a factor of hundreds or even thousands.
- Jason Wang is trying to program an approach based on splitting the tuple. The size is radically larger: $\sim 10^{15}$.