Computing in finitely presented groups

Derek Holt

University of Warwick

Tübingen, September 2018

Contents

Introduction

Computational group theory splits into two not completely disjoint parts: computing in **finite** and **infinite** groups.

In both cases, research is divided into

- **theoretical**: proving decidability and complexity results;
- **practical**: implementing algorithms efficiently.

Introduction

Computational group theory splits into two not completely disjoint parts:

computing in **finite** and **infinite** groups.

In both cases, research is divided into

- **theoretical**: proving decidability and complexity results;
- **practical**: implementing algorithms efficiently.

For computation in finite groups these two approaches are generally compatible and mutually complementary; i.e. in general, algorithms with good complexity have implementations that run faster, at least in large examples.

This is less true for computation in infinite groups, where a polynomial-time algorithm might involve constants that are too large for practical purposes.

Computing in infinite groups

There are two main strands here:

computing in **finitely presented** groups and in **matrix** groups.

Computing in infinite groups

There are two main strands here:

computing in **finitely presented** groups and in **matrix** groups.

Exact computation in infinite groups of matrices (over $\mathbb{Z}$, $\mathbb{Q}$, number fields, function fields, for example) is a relatively new field of research, in which significant advances have been made recently by Flannery, Detinko, O'Brien, Hulpke, et al.

Computing in infinite groups

There are two main strands here:

computing in **finitely presented** groups and in **matrix** groups.

Exact computation in infinite groups of matrices (over $\mathbb{Z}$, $\mathbb{Q}$, number fields, function fields, for example) is a relatively new field of research, in which significant advances have been made recently by Flannery, Detinko, O'Brien, Hulpke, et al.

The input is a finite set of matrices that generate a group G.

Finiteness of G can be decided quickly, as can nilpotency.

The Tits alternative can be decided for G.

But it is unknown whether it is possible to decide whether G is free, even on the given generators.

Computing in finitely presented groups: coset enumeration

From now on, we assume that the group G is defined by a presentation $G = \langle X \mid R \rangle$ with X and R finite. We let $A := X \cup X^{-1}$. So elements of G are represented by words $w \in A^*$.

Computing in finitely presented groups: coset enumeration

From now on, we assume that the group G is defined by a presentation $G = \langle X \mid R \rangle$ with X and R finite. We let $A := X \cup X^{-1}$. So elements of G are represented by words $w \in A^*$.

The **coset enumeration** procedure was formulated by Todd and Coxeter in the 1930s and implemented first in 1953.

The input is X, R, and a finite set $Y \subset A^*$ generating a subgroup $H = \langle Y \rangle$ of G.

If $|G : H|$ is finite, then it, together with the associated action of G on the cosets of H is computed. A presentation of H can also be computed.

So we are effectively computing a **finite quotient** of G, namely the image of this action.

Computing in finitely presented groups: coset enumeration

From now on, we assume that the group G is defined by a presentation $G = \langle X \mid R \rangle$ with X and R finite. We let $A := X \cup X^{-1}$. So elements of G are represented by words $w \in A^*$.

The **coset enumeration** procedure was formulated by Todd and Coxeter in the 1930s and implemented first in 1953.

The input is X, R, and a finite set $Y \subset A^*$ generating a subgroup $H = \langle Y \rangle$ of G.

If $|G : H|$ is finite, then it, together with the associated action of G on the cosets of H is computed. A presentation of H can also be computed.

So we are effectively computing a **finite quotient** of G, namely the image of this action.

Related applications include finding all subgroups of G up to a specified index, so we can systematically enumerate finite quotients of G.

Other quotient algorithms

Other quotient algorithms include algorithms to compute:

- the largest abelian quotient (Smith Normal Form);
- finite p-quotients for a specified prime p;
- nilpotent quotients;
- polycyclic quotients;
- (solvable quotients).

Other quotient algorithms

Other quotient algorithms include algorithms to compute:

- the largest abelian quotient (Smith Normal Form);
- finite p-quotients for a specified prime p;
- nilpotent quotients;
- polycyclic quotients;
- (solvable quotients).

But, unless the group is virtually polycyclic, none of the above techniques enables computations with G itself rather than in a proper quotient of G.

The Dehn Problems

These are decision problems that were formulated by Dehn in 1911. They are basically theoretical questions, but Dehn described implementable algorithms for their solution in some cases.

The Dehn Problems

These are decision problems that were formulated by Dehn in 1911. They are basically theoretical questions, but Dehn described implementable algorithms for their solution in some cases.

- **The Word Problem** WP(G): given $w \in A^*$, is $w =_G 1$?
- **The Conjugacy Problem**: given $v, w \in A^*$, does there exists $c \in A^*$ with $w =_G c^{-1}vc$?
- **The Isomorphism problem**: given another finitely presented group $G' = \langle X' \mid R' \rangle$, is $G \cong G'$?
  (This is the most difficult, both theoretically and practically.)

The Dehn Problems

These are decision problems that were formulated by Dehn in 1911. They are basically theoretical questions, but Dehn described implementable algorithms for their solution in some cases.

- **The Word Problem** WP(G): given $w \in A^*$, is $w =_G 1$?
- **The Conjugacy Problem**: given $v, w \in A^*$, does there exists $c \in A^*$ with $w =_G c^{-1}vc$?
- **The Isomorphism problem**: given another finitely presented group $G' = \langle X' \mid R' \rangle$, is $G \cong G'$?
  (This is the most difficult, both theoretically and practically.)

To this, we can add

- **The Generalized Word Problem** GWP(G, H): given finite $Y \subset A^*$ generating $H = \langle Y \rangle$ and $w \in A^*$, is $w \in H$?

The word problem

Implementations of word problem solutions include:

- **collection** in polycyclic groups;
- the **Dehn algorithm** for hyperbolic groups (linear time);
- putting words into normal form using **finite state automata (fsa)** in **automatic groups** (quadratic time).

The word problem

Implementations of word problem solutions include:

- **collection** in polycyclic groups;
- the **Dehn algorithm** for hyperbolic groups (linear time);
- putting words into normal form using **finite state automata (fsa)** in **automatic groups** (quadratic time).

The class of automatic groups includes: free groups, hyperbolic groups, Coxeter groups, braid groups, mapping class groups, many types of Artin groups, and virtually abelian groups, but **not** other polycyclic groups.

The word problem

Implementations of word problem solutions include:

- **collection** in polycyclic groups;
- the **Dehn algorithm** for hyperbolic groups (linear time);
- putting words into normal form using **finite state automata (**fsa**)** in **automatic groups** (quadratic time).

The class of automatic groups includes: free groups, hyperbolic groups, Coxeter groups, braid groups, mapping class groups, many types of Artin groups, and virtually abelian groups, but **not** other polycyclic groups.

Software for computing with automatic groups includes KBMAG with interfaces to **GAP** and **Magma**, and MAF, by Alun Williams.

Also SNAPPY can be used for 3-manifold groups.

The word problem

Implementations of word problem solutions include:

- **collection** in polycyclic groups;
- the **Dehn algorithm** for hyperbolic groups (linear time);
- putting words into normal form using **finite state automata (**fsa**)** in **automatic groups** (quadratic time).

The class of automatic groups includes: free groups, hyperbolic groups, Coxeter groups, braid groups, mapping class groups, many types of Artin groups, and virtually abelian groups, but **not** other polycyclic groups.

Software for computing with automatic groups includes KBMAG with interfaces to **GAP** and **Magma**, and MAF, by Alun Williams.

Also SNAPPY can be used for 3-manifold groups.

The normal forms most frequently used in programs are **shortlex**: order A and then, for $v, w \in A^*$, defined $v < w$ if either $\ell(v) < \ell(w)$; or $\ell(v) = \ell(w)$ and $v <_{\text{lex}} w$.

Applications of automatic groups software

Once the automatic structure has been computed we can use it to:

- determine whether the group is finite or infinite;
- reduce words quickly (quadratic time) to normal form;
- enumerate normal form words up to a specified length.
- (for geodesic structures) calculate the **growth series** ($\sum_{n\geq 0} c_n x^n$, where c_n is number of group elements with shortest representatives in A^* of length (at most) n) of G as a rational function:

Applications of automatic groups software

Once the automatic structure has been computed we can use it to:

- determine whether the group is finite or infinite;
- reduce words quickly (quadratic time) to normal form;
- enumerate normal form words up to a specified length.
- (for geodesic structures) calculate the **growth series** ($\sum_{n\geq 0} c_n x^n$, where c_n is number of group elements with shortest representatives in A^* of length (at most) n) of G as a rational function:

We have had significant success in resolving some difficult finiteness problems. For example the **Heineken group**

$$\langle x, y, z \mid [[x, [x, y]] = z, [y, [y, z]] = x, [z, [z, x]] = y \rangle$$

is infinite, and even hyperbolic,

as are several difficult cases in the Coxeter family

$$(\ell, m, n; p) = \langle x, y \mid x^\ell, y^m, (xy)^n, [x, y]^p \rangle,$$

such as $(\ell, m, n; p) = (3, 4, 13; 2)$ and $(3, 5, 7; 2)$

as are several difficult cases in the Coxeter family

$$(\ell, m, n; p) = \langle x, y \mid x^\ell, y^m, (xy)^n, [x, y]^p \rangle,$$

such as $(\ell, m, n; p) = (3, 4, 13; 2)$ and $(3, 5, 7; 2)$

Enumerating words has had applications to software for drawing pictures. For example, the vertices in the picture below correspond to elements from the hyperbolic group

$$\langle a, b, c, d, e, f \mid a^4 = b^4 = c^4 = d^4 = e^4 = f^4 = aba^{-1}e = bcb^{-1}f = cdc^{-1}a = ded^{-1}b = efe^{-1}c = faf^{-1}d = 1 \rangle,$$

which is the symmetry group of the tesselation in the picture.

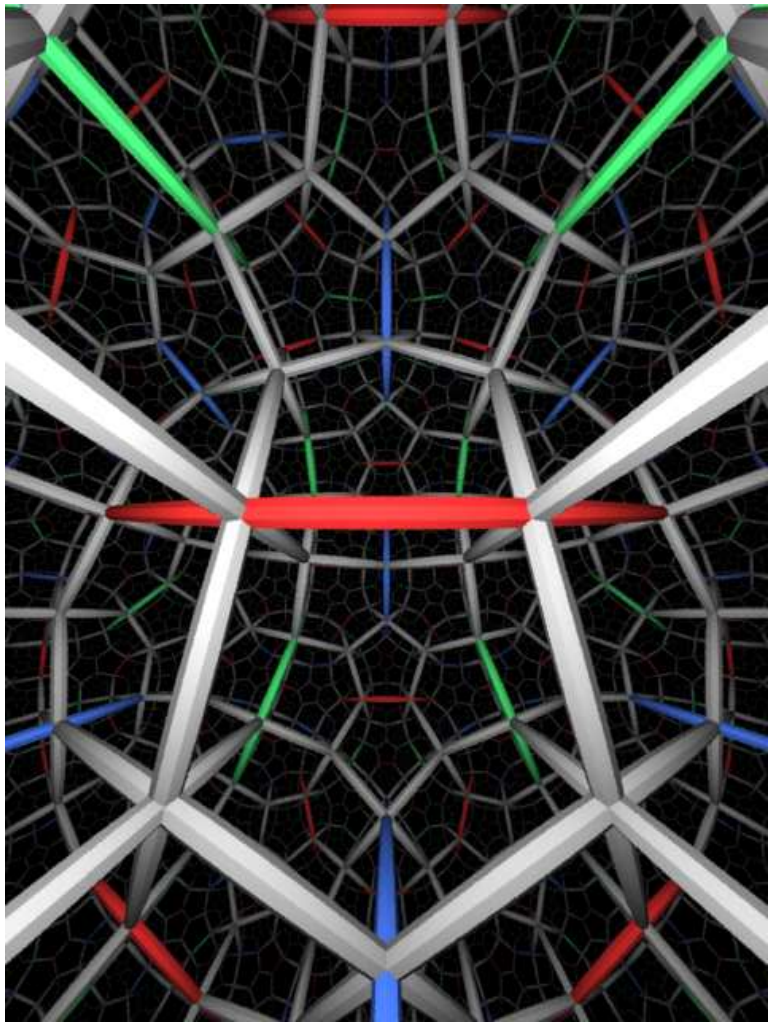


Figure: A tessellation of hyperbolic 3-space by dodecahedra

The generalized word problem: Stallings foldings

Let $G = \langle Y \rangle \leq G$ with $Y \leq A^*$ and Y finite. Recall that the **generalized word problem** $\mathrm{GWP}(G, H)$ is the problem of deciding whether a given $w \in A^*$ lies in H.

The generalized word problem: Stallings foldings

Let $G = \langle Y \rangle \leq G$ with $Y \leq A^*$ and Y finite. Recall that the **generalized word problem** $\mathrm{GWP}(G, H)$ is the problem of deciding whether a given $w \in A^*$ lies in H.

The **Stallings folding** method is a linear-time algorithm for solving this problem when G is free. A fsa is constructed that accepts a *reduced* word in $w \in A^*$ if and only if $w \in H$.

The generalized word problem: Stallings foldings

Let $G = \langle Y \rangle \leq G$ with $Y \leq A^*$ and Y finite. Recall that the **generalized word problem** GWP(G, H) is the problem of deciding whether a given $w \in A^*$ lies in H.

The **Stallings folding** method is a linear-time algorithm for solving this problem when G is free. A fsa is constructed that accepts a *reduced* word in $w \in A^*$ if and only if $w \in H$.

The folding process is essentially the same as coincidence processing in coset enumeration, and it can be done that way using existing implementations with correct parameter setting.

The best modern implementation of coset enumeration is probably the **ACE** package by Havas and Ramsay, which can be accessed from both **GAP** and **Magma**.

Example

Example

```
gap> LoadPackage("ACE");;
gap> G := FreeGroup(2);;
gap> gens := [G.1, G.2];;
gap> subgens := [G.1^3, G.2^3, (G.1*G.2)^3];;
gap> ACECosetTableFromGensAndRels(gens,[],subgens :
                                        purer,incomplete);
I  ACECosetTable: Coset table is incomplete,
                            reduced & lenlex standardised.
  [ [ 2, 3, 1, 0, 0, 8, 5, 0 ],     #G.1
    [ 3, 1, 2, 0, 7, 0, 0, 6 ],     #G.1^-1
    [ 4, 6, 0, 5, 1, 0, 0, 7 ],     #G.2
    [ 5, 0, 0, 1, 4, 2, 8, 0 ] ]    #G.2^-1
```

Quasiconvex subgroups

Ilya Kapovich (1997) generalized Stallings foldings to other classes of groups.

A subgroup $H \leq G$ is **quasiconvex** if geodesics in the Cayley graph for G that begin and end at vertices in H remain uniformly close to H. For example, all finitely generated subgroups of a finitely generated free group are quasiconvex.

Quasiconvex subgroups

Ilya Kapovich (1997) generalized Stallings foldings to other classes of groups.

A subgroup $H \leq G$ is **quasiconvex** if geodesics in the Cayley graph for G that begin and end at vertices in H remain uniformly close to H. For example, all finitely generated subgroups of a finitely generated free group are quasiconvex.

For a language $L \subseteq A^*$ that maps onto G, we say that H is **L-quasiconvex** in G if paths labelled by words in L that begin and end in H remain uniformly close to H.

In that case, there is an fsa W_H that accepts words in L if and only if they lie in H. (We can take W_H to be a neighborhood of the origin in the Schreier graph for H in G.) So words in normal form L can be tested for membership in H in linear time.

In particular, if G is automatic, then we can take L to be its associated language. Then words can be put into normal form L in quadratic time.

In particular, if G is automatic, then we can take L to be its associated language. Then words can be put into normal form L in quadratic time.

We can attempt to construct W_H as follows.

First find a candidate $\overline{W}_H$ for W_H with $L(\overline{W}_H) \subseteq L(W_H)$. Then

$$L(\overline{W}_H) = L(W_H) \iff$$
$$\forall \left(w \in L(\overline{W}_H),\, u \in Y^{\pm},\, v \in L \text{ s.t. } v =_G wu \right) : v \in L(W_H).$$

This condition can be tested and, if it fails, words $w \in L(W_H) \setminus L(\overline{W}_H)$ are found, which enable an improved candidate $\overline{W}_H$ to be constructed.

In particular, if G is automatic, then we can take L to be its associated language. Then words can be put into normal form L in quadratic time.

We can attempt to construct W_H as follows.

First find a candidate $\overline{W}_H$ for W_H with $L(\overline{W}_H) \subseteq L(W_H)$. Then

$$L(\overline{W}_H) = L(W_H) \iff$$
$$\forall \left(w \in L(\overline{W}_H),\, u \in Y^{\pm},\, v \in L \text{ s.t. } v =_G wu \right) : v \in L(W_H).$$

This condition can be tested and, if it fails, words $w \in L(W_H) \setminus L(\overline{W}_H)$ are found, which enable an improved candidate $\overline{W}_H$ to be constructed.

The initial candidate $\overline{W}_H$ can be found by using coset enumeration or Knuth-Bendix to construct part of the Schreier graph.

In particular, if G is automatic, then we can take L to be its associated language. Then words can be put into normal form L in quadratic time.

We can attempt to construct W_H as follows.

First find a candidate $\overline{W}_H$ for W_H with $L(\overline{W}_H) \subseteq L(W_H)$. Then

$$L(\overline{W}_H) = L(W_H) \iff$$
$$\forall \left(w \in L(\overline{W}_H),\, u \in Y^{\pm},\, v \in L \text{ s.t. } v =_G wu \right) : v \in L(W_H).$$

This condition can be tested and, if it fails, words $w \in L(W_H) \setminus L(\overline{W}_H)$ are found, which enable an improved candidate $\overline{W}_H$ to be constructed.

The initial candidate $\overline{W}_H$ can be found by using coset enumeration or Knuth-Bendix to construct part of the Schreier graph.

Alternatively, we can try to find a **strong coset automatic system** for (G, H) (Holt/Hurt, 1999). This enables words $w \in A^*$ to be reduced in quadratic time to a canonical representative of their coset - typically the shortlex least representative. This can be used to construct the required portion of the Schreier graph accurately.

Applications of coset automatic systems software

Analogously to the automatic structures the coset automatic system can be used to:

- determine whether $|G : H|$ is finite or infinite;
- reduce words quickly (quadratic time) to a normal form for their coset;
- enumerate normal form words up to a specified length.

An application of word enumeration to drawing pictures is described in the paper:

G. McShane, J. Parker and I. Redfern, "Drawing Limit Sets of Kleinian Groups Using Finite State Automata", Experimental Mathematics, **3** (1994), 153–172.

The conjugacy problem

There are groups, including some subgroups of $F_2 \times F_2$, with solvable word problem but unsolvable conjugacy problem

The conjugacy problem

There are groups, including some subgroups of $F_2 \times F_2$, with solvable word problem but unsolvable conjugacy problem

There are also groups, such as braid groups and polycyclic groups for which the conjugacy problem is solvable but appears to be significantly more difficult than the word problem.

Some protocols for cryptosystems that utilise this situation have been proposed, but the resulting research led to substantial improvements in methods for the braid group, so they are unlikely to be used in pracrice.

The conjugacy problem

There are groups, including some subgroups of $F_2 \times F_2$, with solvable word problem but unsolvable conjugacy problem

There are also groups, such as braid groups and polycyclic groups for which the conjugacy problem is solvable but appears to be significantly more difficult than the word problem.

Some protocols for cryptosystems that utilise this situation have been proposed, but the resulting research led to substantial improvements in methods for the braid group, so they are unlikely to be used in pracrice.

Dehn proposed a solution for surface groups, which was generalized by Greendlinger to $C'(1/8)$ small cancellation groups.

These algorithms are easily implemented, but are quadratic time, and involve considering all cyclic conjugates of both input words, so they become slow when testing long words.

The conjugacy problem in hyperbolic groups

A quadratic time algorithm for hyperbolic groups is described in the book by Bridson and Haefliger.

This was improved to linear time by Holt and Epstein in 2006. But unfortunately these methods are not practical, and typically involve trying an astronomically large number of possible conjugating elements.

The conjugacy problem in hyperbolic groups

A quadratic time algorithm for hyperbolic groups is described in the book by Bridson and Haefliger.

This was improved to linear time by Holt and Epstein in 2006. But unfortunately these methods are not practical, and typically involve trying an astronomically large number of possible conjugating elements.

A practical method for testing non-torsion elements for conjugacy in hyperbolic groups was described and implemented by **Joe Marshall** in 2002. This, together with related algorithms, including a test for malnormality of quasiconvex subgroups of hyperbolic groups, were based on ideas of **Eric Swenson**.

The conjugacy testing has recently been re-implemented in **GAP**, and is available on request.

Straight elements

Definition

If $G = \langle X \rangle$, then a word $w \in A^*$ is **straight** if all powers w^m of w with $m > 0$ are geodesic words. An element $g \in G$ is **straight** if it is represented by a straight word.

Straight elements

Definition

If $G = \langle X \rangle$, then a word $w \in A^*$ is **straight** if all powers w^m of w with $m > 0$ are geodesic words. An element $g \in G$ is **straight** if it is represented by a straight word.

Theorem

If G is hyperbolic and $g \in G$ has infinite order, then some power g^m of g is conjugate to a straight element.

Furthermore, there is an upper bound on m that depends only on G and X.

Conjugacy testing of non-torsion elements

Suppose that we want to test the non-torsion elements $g_1, g_2 \in G$ for conjugacy in G, where G is hyperbolic.

The first step in the Swenson/Marshall algorithm is to find conjugates

$$h_1 = c_1 g_1 c_1^{-1} \text{ and } h_2 = c_2 g_2 c_2^{-1}$$

of g_1 and g_2 such that h_1^m and h_2^m are straight for some $m > 0$.

Conjugacy testing of non-torsion elements

Suppose that we want to test the non-torsion elements $g_1, g_2 \in G$ for conjugacy in G, where G is hyperbolic.

The first step in the Swenson/Marshall algorithm is to find conjugates

$$h_1 = c_1 g_1 c_1^{-1} \text{ and } h_2 = c_2 g_2 c_2^{-1}$$

of g_1 and g_2 such that h_1^m and h_2^m are straight for some $m > 0$.

Our current implementation of this step in **GAP** is naive, and is not guaranteed to find the smallest possible m, but in practice m is usually moderately small (at most 10). This is important because, as we shall see, smaller values of m improve the efficiency of the conjugacy test.

Conjugacy testing of non-torsion elements

Suppose that we want to test the non-torsion elements $g_1, g_2 \in G$ for conjugacy in G, where G is hyperbolic.

The first step in the Swenson/Marshall algorithm is to find conjugates

$$h_1 = c_1 g_1 c_1^{-1} \text{ and } h_2 = c_2 g_2 c_2^{-1}$$

of g_1 and g_2 such that h_1^m and h_2^m are straight for some $m > 0$.

Our current implementation of this step in **GAP** is naive, and is not guaranteed to find the smallest possible m, but in practice m is usually moderately small (at most 10). This is important because, as we shall see, smaller values of m improve the efficiency of the conjugacy test.

So g_1 and g_2 are conjugate with $dg_1 d^{-1} = g_2$ if and only if $ch_1 c^{-1} = h_2$, with $c = c_2 d c_1^{-1}$, in which case $ch_1^n c^{-1} = h_2^n$ for all $n \geq 0$.

Geodesic rays

In a hyperbolic group, any two infinite rays in the Cayley graph that start at the base point either diverge exponentially or fellow-travel.

Starting from the automatic structure of G, we can compute a 2-tape fsa with language

$$\{(w_1(k), w_2(k)) : k \geq 0,\ w_1, w_2 \text{ fellow-travelling geodesic rays } \}.$$

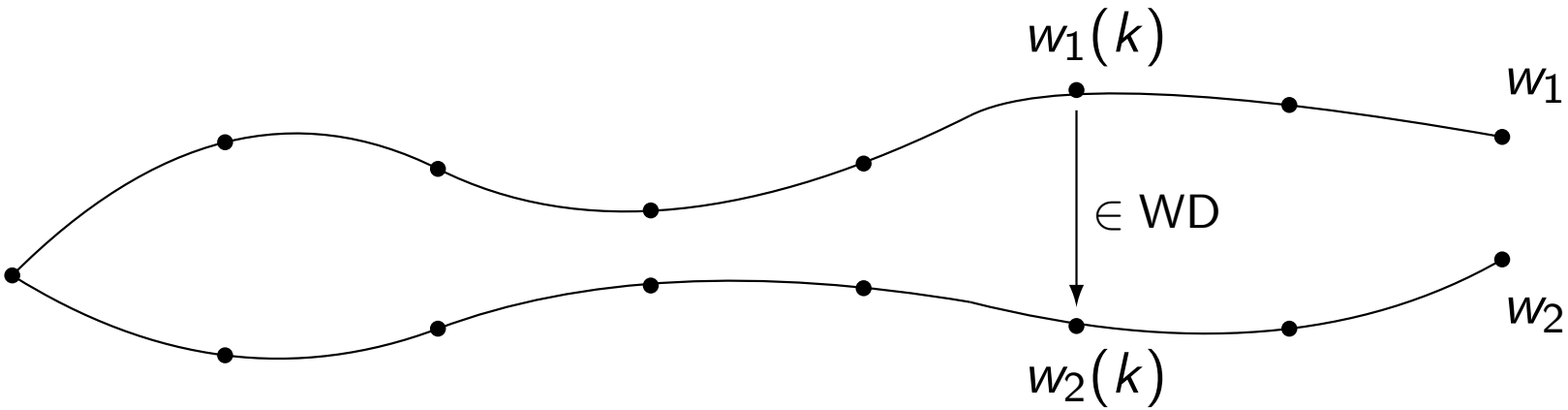


Figure: Fellow-travelling geodesic rays w_1, w_2

Conjugacy testing of non-torsion elements (ctd)

From this fsa, we can compute the set WD of associated word-differences

$$\{w_1(k)^{-1}w_2(k) \in G : k \geq 0,\ w_1, w_2 \text{ fellow-travelling geodesic rays } \}.$$

Conjugacy testing of non-torsion elements (ctd)

From this fsa, we can compute the set WD of associated word-differences

$$\{w_1(k)^{-1}w_2(k) \in G : k \geq 0,\ w_1, w_2 \text{ fellow-travelling geodesic rays } \}.$$

Now let w_1 and w_2 be the shortlex straight representatives of h_1^m and h_2^m. So w_1 and w_2 are straight.

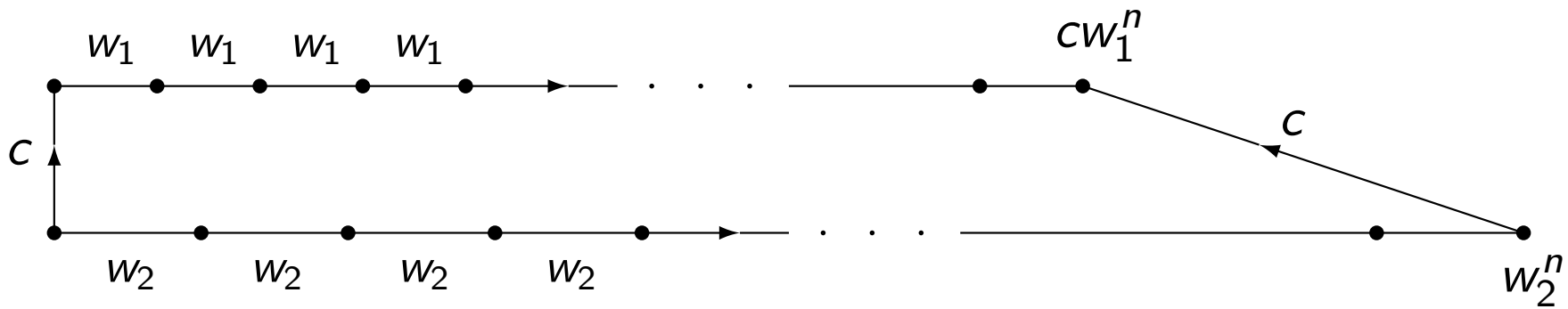
Conjugacy testing of non-torsion elements (ctd)

From this fsa, we can compute the set WD of associated word-differences

$$\{w_1(k)^{-1}w_2(k) \in G : k \geq 0,\ w_1, w_2 \text{ fellow-travelling geodesic rays } \}.$$

Now let w_1 and w_2 be the shortlex straight representatives of h_1^m and h_2^m. So w_1 and w_2 are straight.

Suppose that g_1 and g_2 are conjugate. Then there exists c with $ch_1c^{-1} = h_2$ and so $cw_1^n =_G w_2^n c$ for all $n \geq 0$.



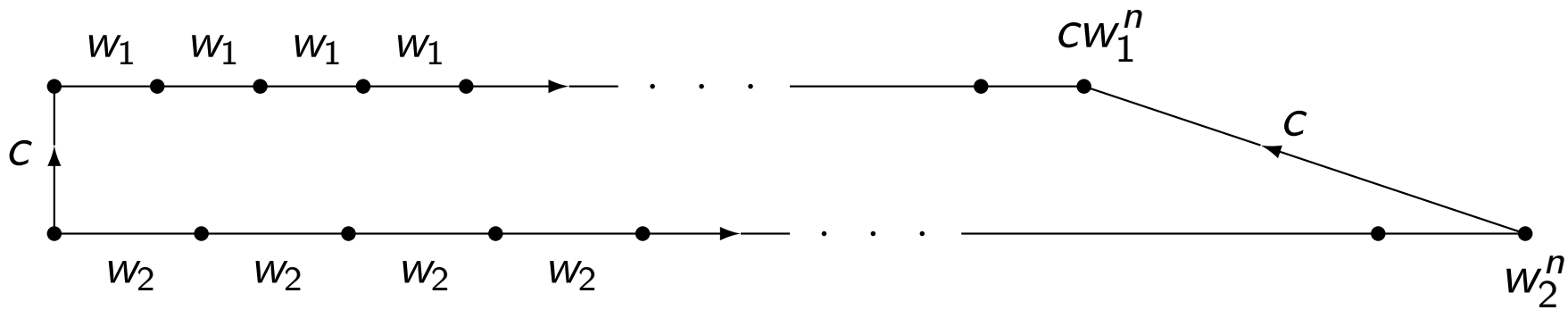
Conjugacy testing of non-torsion elements (ctd)

From this fsa, we can compute the set WD of associated word-differences

$$\{w_1(k)^{-1}w_2(k) \in G : k \geq 0,\ w_1, w_2 \text{ fellow-travelling geodesic rays } \}.$$

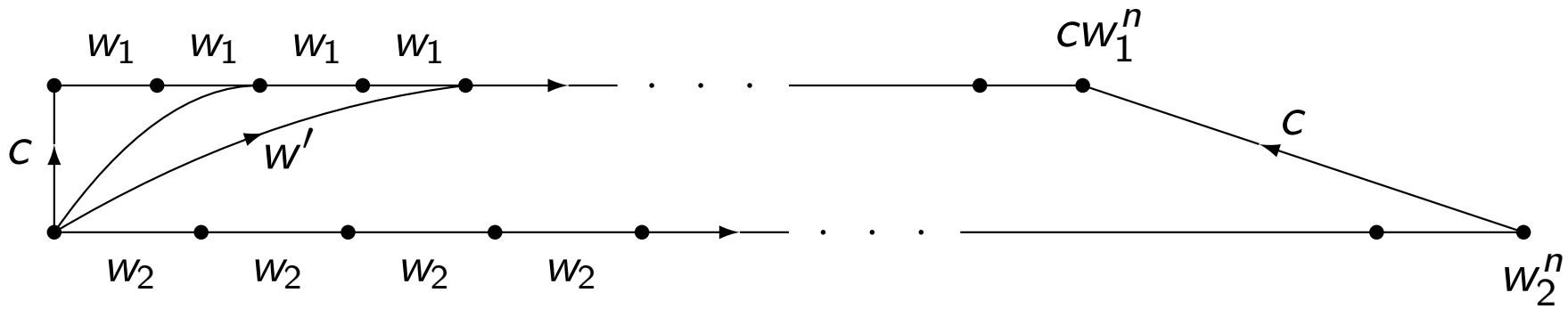
Now let w_1 and w_2 be the shortlex straight representatives of h_1^m and h_2^m. So w_1 and w_2 are straight.

Suppose that g_1 and g_2 are conjugate. Then there exists c with $ch_1c^{-1} = h_2$ and so $cw_1^n =_G w_2^n c$ for all $n \geq 0$.

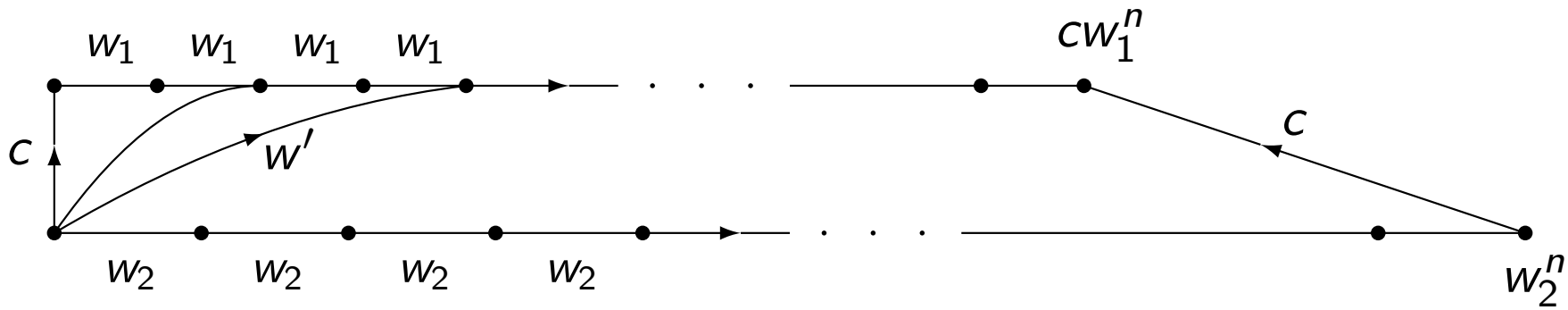


Then, for some $\ell \geq 0$, the words $w'w_1^n$ are geodesics for all $n \geq 0$, where w' is a geodesic word for cw_1^ℓ.

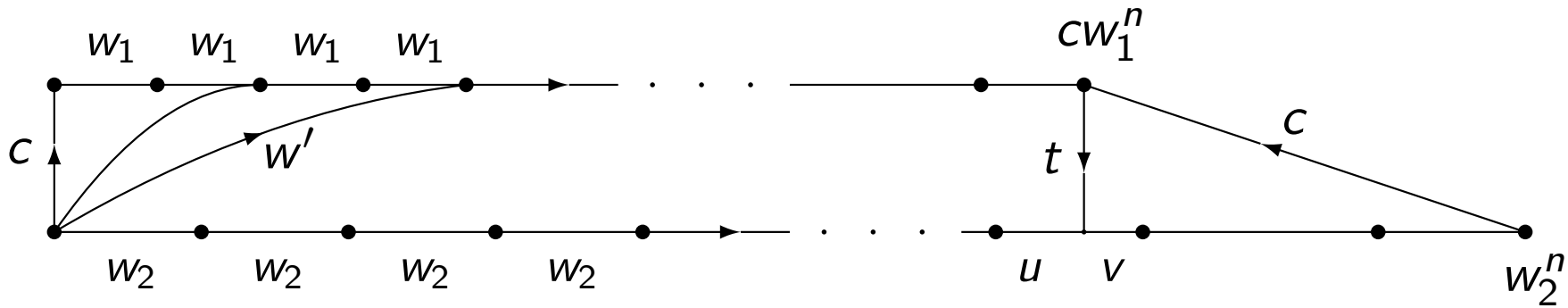
So $w'w_1^\infty$ and w_2^∞ are fellow-travelling infinite geodesic rays:



So $w'w_1^\infty$ and w_2^∞ are fellow-travelling infinite geodesic rays:



So there is a $t \in \mathrm{WD}$ and a suffix v of w_2 such that $c^{-1} =_G tvw_2^k$ for some $k \in \mathbb{Z}$.



Then, since $w_2 =_G h_2^m$, we have

$$dg_1d^{-1} = g_2 \iff ch_1c^{-1} = h_2 \iff (tv)^{-1}h_1tv = h_2.$$

Then, since $w_2 =_G h_2^m$, we have

$$dg_1d^{-1} = g_2 \iff ch_1c^{-1} = h_2 \iff (tv)^{-1}h_1tv = h_2.$$

So to test g_1 and g_2 for conjugacy, it is sufficient to test the condition $(tv)^{-1}h_1(tv) =_G h_2$ for all $t \in \mathrm{WD}$ and for all suffixes v of w_2.

The number of suffixes of w_2 is equal to the length $\ell(w_2)$ of w_2, which is $m\ell(g_2)$. So it is desirable to choose m as small as possible. Theoretically it is bounded as a function of the presentation of G.

Since solving the word problem is quadratic time in our implementation, the complete process has cubic time complexity.

Example

The following example is the first group in the Hodgson-Weeks database of fundamental groups of hyperbolic 3-manifolds.

Example

The following example is the first group in the Hodgson-Weeks database of fundamental groups of hyperbolic 3-manifolds.

Example

$$G = \langle a, b \mid ababa^{-1}b^2a^{-1}b, abab^{-1}a^2b^{-1}ab \rangle$$

```
Word acceptor, geodesic word acceptor:  1067, 1115 states.
Geodesic rays acceptor: 3711 states, |WD| = 141.
Powers to get straight conjugates: 1 (usually), 2, 3, 5.
Time for 700 conjugacy tests of words of length 6:  120 seconds
Time for one conjugacy test of words of length 200:  5.4 seconds
Time for one conjugacy test of words of length 600:  42 seconds
```

Some challenges

Find effective methods for implementing algorithms in geometric group theory that have been proved to be theoretically efficient. For example:

Some challenges

Find effective methods for implementing algorithms in geometric group theory that have been proved to be theoretically efficient. For example:

Some challenges

Find effective methods for implementing algorithms in geometric group theory that have been proved to be theoretically efficient. For example:

- The conjugacy problem for torsion elements of hyperbolic groups.

Some challenges

Find effective methods for implementing algorithms in geometric group theory that have been proved to be theoretically efficient. For example:

- The conjugacy problem for torsion elements of hyperbolic groups.
- The **compressed word problem** in free groups or (more generally) in hyperbolic groups G. This would lead to effective algorithms for the word problem in $\mathrm{Aut}(G)$.
  For example, for $a \in G$, Then $(a, a_1 = a^2, a_2 = a_1^2, \ldots, a_n = a_{n-1}^2)$ is a compressed word (or straight line program) for $a_n = a^{2^n}$.

Some challenges

Find effective methods for implementing algorithms in geometric group theory that have been proved to be theoretically efficient. For example:

- The conjugacy problem for torsion elements of hyperbolic groups.
- The **compressed word problem** in free groups or (more generally) in hyperbolic groups G. This would lead to effective algorithms for the word problem in $\mathrm{Aut}(G)$.
  For example, for $a \in G$, Then $(a, a_1 = a^2, a_2 = a_1^2, \ldots, a_n = a_{n-1}^2)$ is a compressed word (or straight line program) for $a_n = a^{2^n}$.
- Can we deal with infinite families of groups, perhaps using a combination of computation and hand calculations?

Some challenges

Find effective methods for implementing algorithms in geometric group theory that have been proved to be theoretically efficient. For example:

- The conjugacy problem for torsion elements of hyperbolic groups.
- The **compressed word problem** in free groups or (more generally) in hyperbolic groups G. This would lead to effective algorithms for the word problem in $\mathrm{Aut}(G)$.
  For example, for $a \in G$, Then $(a, a_1 = a^2, a_2 = a_1^2, \ldots, a_n = a_{n-1}^2)$ is a compressed word (or straight line program) for $a_n = a^{2^n}$.
- Can we deal with infinite families of groups, perhaps using a combination of computation and hand calculations?

Thank you for listening!